Thai Food Image Recognition Models built from TensorFlow and Keras

Tanaporn Rojanaridpiched and Tanawin Wichit
Faculty of Information and Communication Technology, Mahidol University, Nakhon Pathom, 73170, Thailand
Email: tanaporn.roj, tanawin.wic@student.mahidol.ac.th

Abstract—Due to the nature of cooking and food in general, recognizing foods from their visual appearance proves to be a difficult task because of the visual diversity and the visual similarity of each type of food. In this paper, we conducted a comparative study on multiple deep learning-based models for the multi-class Thai food recognitions using food dish photos. We experimented with a total of 8 classification models, half of which are designed and trained from scratch. The other half is built upon existing pre-trained models using transfer learning. To evaluate the performance of all models, we also accumulated photos from the internet and our own photos to create a dataset of 38 classes each of which has 250 images. We discovered that models from transfer learning are generally more efficient in training and more performant even when the dataset is small than the ones trained from scratch. Moreover, we also found that training models from scratch are time-consuming and troublesome because there are countless hyperparameters to be dealt with. However, despite the fact that transfer learning models are better, they are unable to distinguish and correctly classify some distinct kind of foods are visually similar.

Index Terms— Thai Food Image Processing, Thai Food Image Recognition, Deep Learning, Convolutional Neural Network (CNN), Computer Vision, Transfer Learning, TensorFlow, Keras, Python

I INTRODUCTION

Food is essential for all humans and, in fact, all living organisms because it provides the energy that it needs to function. Furthermore, for humans, food is also an ultimate indicator of culture, belief, geological characteristics, and even flora of a country. Foods in Thailand are well-known for their variety, tastes, and ingredients. For most foreigners and some Thai people, remembering and identifying local menus and dishes proves to be troublesome since there are too many variations of the same food depending on the region and the individual. In fact, this is because there is no absolute correct method to cook any food. Furthermore, misidentifying foods may lead to miscommunication between persons especially when ordering food from a made-to-order bistro that can be commonly found in Thailand. Therefore, misordering foods from Thai restaurants could further cause unnecessary money spending, dissatisfaction, or even allergic symptom. To solve such a problem, conventionally, people usually have to become observant of ingredients, visual appearance, and odor of each type of food. Doing so requires an extensive period of time to master or get familiar with. On the contrary, with the advancement and extensive use of computer vision-related tasks, technology has become widely popular for food image recognition. This is because the integration of computer vision usually provides an easy-to-understand experience for end-user of any applications. To overcome this difficulty of manual food recognition while providing good experiences for users, building an automatic system, that can identify food from a photograph, using highly-abundant computing resources would be a logical regime to invest resource into.

Therefore, this study tries to experiment with various popular sophisticated deep learning models with convolution neural networks (CNN) for Thai food image recognition. The goal is to explore the best model and configurations that can accurately identify the food name when a food image is provided. We will use well-establish empirical methods to do so. Moreover, before start experimenting, we will also gather images, that contain food that is commonly found near the Salaya campus of Mahidol University. Each of the images in the dataset was taken from different lightning conditions and angles, to form a dataset that will be used to train models that can tolerate noises in the input image.

II LITERATURE REVIEW

In the context of food, computer vision, and image processing, recognizing food images usually is the task that is widely discussed because it is the foundation for further applications, that require food name labels. Some of the examples would be food calorie estimation based on a given image of food. If the name of the food is known, a system can, at minimum, trivially estimate the number of calories of the food. The given application example is highly useful practically and commercially as a health-conscious mindset spread amongst Thai society. Currently, to perform image recognition in general, there are two general approaches as follows: machine learning-based techniques and deep learning-based techniques.

A Machine Learning-based Food Recognition

One of the most popular ML techniques that work well for food image recognition is Support Vector Machine (SVM). Pouladzadeh, *et al.* [1] proposed an approach for SVM food classification for further food calorie estimation. Shape, color, size, and texture characteristics are extracted and used as features for classification.

However, Yoshiyuki Kawano and Keiji Yanai [2] also worked on a food classification task with a different feature extraction approach using a linear SVM classifier with X^2 Kernel. They make use of foreground extraction, color histogram, and bag-of-features for the feature extraction process. The implementation proves to be performant and efficient enough to run on a low-powered CPU on an Android phone in real-time recognition.

Pouladzadeh, *et al.* [3] also purposed an SVM classifier with Radial Basis Function (RBF) kernel that takes food size, shape, color, and texture features. They introduced a method to automatically determine the area and volume of foods in a photo. Users are required to take 2 images with different angles and their thumb finger in the frame of the same food. After the food is recognized, along with area and volume information, the number of calories can be estimated. The used dataset takes uncertainty measurements into account by getting images of a dish of food in different illuminations, camera setup, and angle. The research shows a promising accuracy in terms of approximating food weight and calories.

Moreover, besides the SVM approach, the Bayesian framework is also being adopted for food recognition. Aizawa, *et al.* [4] creates an application that aims to provide dietary insight by allowing its users to upload, analyze, and record photos. Furthermore, they conducted a comparison between a deterministic approach and a non-deterministic approach. The deterministic approach uses SVM for both food image detection and food recognition tasks. Features from a food image are extracted in terms of color, circular shape, a bag of features, and block features. In contrast, for the non-deterministic approach, the researchers use the Naive Bayes classifier for both tasks. The comparison shows that both methods are yield a similar F1 score. Furthermore, they also conducted a comparative experiment on 2 types of the Naive Bayes classifiers: a global classifier for every user, and personalized classifier for each user. The result shows that the latter method yields better accuracy. One observation that could be made is that each research related to SVM-based classifiers to solve a similar problem tends to differ in terms of problem modeling, particularly, the feature extraction process.

B Deep Learning-based Food Recognition

Another approach that is frequently used for the food image classification task is a deep neural network (DNN).

Kagaya, *et al.* [5] conducted comparative studies between a certain type of deep neural network named Convolutional Neural Network (CNN) and SVM to detect and recognize food images. Furthermore, after observing from convolution kernels, the researcher discovered that the most important feature for food detection is color features. The result is that CNN performs significantly better than SVM with handcrafted features in classification accuracy metrics.

Unlike several studies that assume one food per image, Aslan, *et al.* [6] tried to recognize foods, and estimate their quantity and calories by segmenting regions of images that are food via CNN. They assumed each food image may have multiple kinds of foods. Data augmentation techniques such as applying brightness changes, JPEG compression artifact, Gaussian noise, white balance, and Gaussian blur to the dataset is used to mimic real-world images where an environmental factor is highly variable. They visualized how noises in images can obstruct the learning algorithm to identify proper food segmentation. The result is the original image can be detected better than a blur and noisy image. They found that SSNet is the best segmentation algorithm.

Pouladzadeh, *et al.* [7] tried to speed up and improve the accuracy of the classification algorithm to make it viable on a low-powered mobile central processing unit (CPU) of most smartphones. They used Region Proposal Algorithm (RPA) to infer regions of interest. After

that, each region undergoes the process of CNN feature extraction and region mining. In the end, they successfully achieve fast response time because heavy computational tasks can be performed locally on the user's device.

Similarly to the previous study, Termritthikun, *et al.* [8] realized that the processing time and the amount of the parameters (or model size) should be reduced. So they tried to reduce processing time and the model size so that the program can be used with smartphones. They found NU-InNet recognition's precision to the same level as GoogLeNet's but the processing time and the model size is reduced because they specifically adjusted hyperparameters such as layers to suit their dataset.

Ciocca, *et al.* [9] proposed a new dataset for food recognition algorithms evaluation and machine learning-based techniques: SVM and k-NN that used CNN-based features to learn upon. Similar to [5], food segmentation is performed but via a traditional image transformation pipeline. For each region-of-interest, features are extracted globally and locally via a patch-based method.

Zhang, *et al.* [10] experimented with a multi-task system that can identify dish types, food ingredients, and cooking methods from food images with deep CNN. They reduced the noises of the data, which was collected from the Internet, outlier images were detected and eliminated through a one-class SVM trained with deep convolutional features. The proposed framework shows a higher accuracy than the traditional ML method.

Fig. 1: An illustration of intra-class variability of food appearances. Left image pair is classified as "Steak", while the right pair represents "Rice topped with stir-fried meat and basil."

Common challenges of the food image classification task are the high intra-class variability. In other words, the very same dish of food may have a visual appearance similar to that of a dish of food from another class as shown in Fig. 1. This is because identical kinds of food can have distinct ways of preparations and ingredients as well as the condition in which food image is captured. For example, in an environment with a warm temperature light setup, foods may appear indistinguishable from others due to an altered color appearance. On the contrary, low inter-class variability is another challenge where some foods may appear nearly identical to other types of foods. Figure 2 illustrates the later problem.

Fig. 2: An illustration of low inter-class variability of food appearances. The top row shows the strike similarity between "Green Curry" and "Yellow Curry." The bottom row shows another similarity between "Fried Mussel Pancakes" and "Omelette."

III METHODOLOGY

In this section, the used datasets, the structure, and the design for the proposed Thai food image recognition system will be discussed in detail.

A Data Gathering

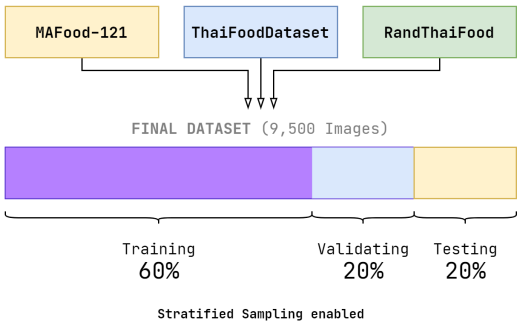


Fig. 3: An overview illustration describes how the dataset is composed and used for training, validation, and testing procedure.

In our experiment, we have gathered a handful of datasets that contain foods commonly found in Thailand to use it for the training and testing process of food image classifiers. This section explains the characteristics of each dataset that we have chosen.

Multi-Attribute Food 121 (MAFood-121) [11] is a

Fig. 4: Sample food images from MAFood-121 dataset

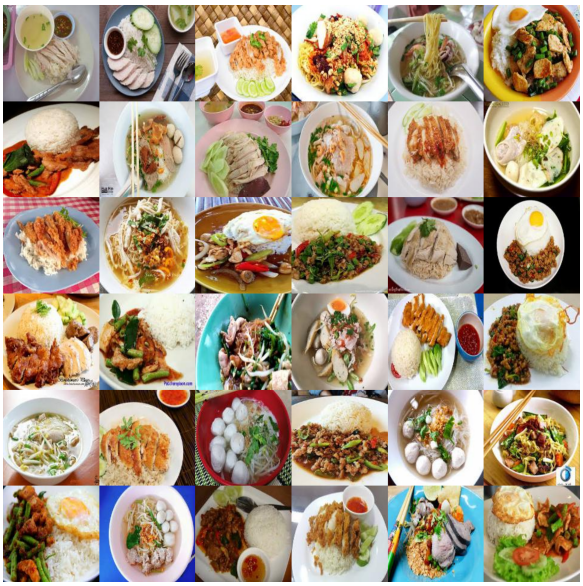


Fig. 5: Sample food images from Thai Food Photo Recognition Dataset dataset (dubbed as ThaiFoodRecog)

dataset comprised of 21,175 size-varied RGB-colored food images that can be further categorized into 121 types of foods from US, China, France, Greece, India, Italy, Japan, Mexico, Thailand, Turkey, and Vietnam. The dataset was created as part of multi-tasks food recognition models by Eduardo, *et al.* in 2019. Furthermore, besides food images for food dish name recognition, the dataset also includes labels or ground truth for every image in terms of cuisine as well as food categories such as bread, egg, meat, etc. For the Thai food recognition task, images from certain classes in the dataset will be used due to the fact that, as mentioned, there are foods that are not relevant to the scope of the study. Figure 4 shows some of the images that are included with the dataset.

Thai Food Photo Recognition Dataset by Ukrit Tiankaew and Peerapon Chunpongthong [12] is a dataset composed of totally 13 types of commonly discoverable Thai foods such as Fried Pork with Garlic Pepper on Rice, etc. Originally, the dataset has no unique name; therefore, we dubbed it *ThaiFoodRecog*. Each type of food has approximately 400 160-by-160-pixels RGB images that are collected from the internet. Additionally, images in the dataset also have a considerable amount of variance due to the fact that there is a significant amount of variation for the image quality that is the result of the lack of controlled environments for a uniform lighting condition, background color, and shadow. Lastly, images in the dataset were upscaled to 640-by-640 pixels by using free software. Figure 5 shows sample images from the dataset.

On top of the aforementioned datasets discovered on the internet, our team also put a tremendous effort to capture dishes of Thai foods commonly found within or nearby Mahidol University located at Salaya, Nakorn Pathom province, Thailand. We called the dataset *ThaiRandFoods*. Our dataset is comprised of 38 classes of food each has 250 images. Some of the images are captured using various models of modern smartphones spread amongst 23 classes out of the 38 which include Caesar Salad, Fried chicken with rice, Noodles, etc. On top of manual capturing, the dataset also consists of images that were taken from the internet through Google Image search. Each image taken was handpicked and cropped to ensure the reliability of the dataset. Despite having put efforts to maintain consistency, having foods in a controlled environment where lighting, white balance, and shadow are assimilative proved to be a difficult task to achieve practically because each food image is captured before it was consumed. As a consequence, images in our dataset do unavoidably have a wide variety of light conditions, shadows, and backgrounds that may or may not affect classification performance. Figure 6 illustrates sample images from our dataset.

All of the 3 datasets will be combined together to form the final dataset with a total of 38 classes each with 250 classes. Totally there are 9500 images. Please note that mismatched and irrelevant classes are removed. The reason why we are using external datasets is that we

Fig. 6: Sample food images from our own dataset named *ThaiRandFoods*

have limited resources and time to collect an entirely new dataset.

B Data Segmentation

In this literature, as illustrated in figure 3, the combined dataset will be segmented with deterministic stratified sampling into 3 portions: training, validation, and testing. The portions have 60%, 20%, and 20% respectively. This is to ensure that the trained model is not biased to any of the 38 classes. This is to avoid the model being fitted to only a few classes.

C Data Preprocessing & Augmentation

Each image will undergo an image resizing process to make all images compatible with the 256-by-256 input layer. Moreover, after the image is resized, data augmentation is deployed in order to make certain that the models make models learn transformation-invariant features. Moreover, data augmentation is used to increase the diversity of the dataset images to ensure that the model can generalize to unseen data. Therefore, the augmentations that are used are geometric-cosmetic transformations such as horizontal and vertical flips, rotation, and zoom as illustrated in figure 7. This augmentation is applied randomly to different images. Consequently, before the forward propagation beings, every image will have its pixel color values rescaled to a proper range such as $[0, 1]$ depending on a model that is involved.

Fig. 7: Possible augmentations that are being made to one of images from the *steam rice roll* class.

D Classification

In this project, we will experiment with X models both comprised transfer learning using pre-trained models and training from scratch. This is to explore which regime should be invested in when creating a deep learning-based classifier. Table 1 illustrates the composition of our experiments.

As illustrated in figure 8, regardless of whether it is training or not, when an image of any arbitrary size is provided, it will be resized to 256-by-256 pixels while retaining its RGB color channels. After get resized, the image will be randomly transformed as part of the previously mentioned data augmentation. The augmented image will later be passed to the model that has CNN as the main feature extractor and the fully connected layer as the classifier. CNN will begin its forward propagation process to get crucial features out of the image. Lastly, using the feature extracted by the CNN, the fully connected layer will return a vector of the probability of being each of 38 classes with the help of Softmax. Please also note that the label of each image is one-hot encoded.

E Training from scratch method

The workflow when training from scratch starts after the data is split into 3 portions as mentioned. There are 2 major phases as follows:

1. **Training Phase** - After the model is defined, the training data will be fed to the model for training. During this, the model will be segmented into small batches for faster parameter updates. Forward and backward propagation will repeatedly

Table 1: MODELS USED IN THE EXPERIMENT

Model name	Method	No. of Parameters
Custom 1	Training from scratch	1,851,914
Custom 2	Training from scratch	2,651,954
Custom 3	Training from scratch	1,943,002
Custom 4	Training from scratch	3,272,122
DenseNet201	Transfer learning	18,524,686
ResNet101v2	Transfer learning	42,842,062
ResNet50v2	Transfer learning	23,778,634
Xception	Transfer learning	21,142,921

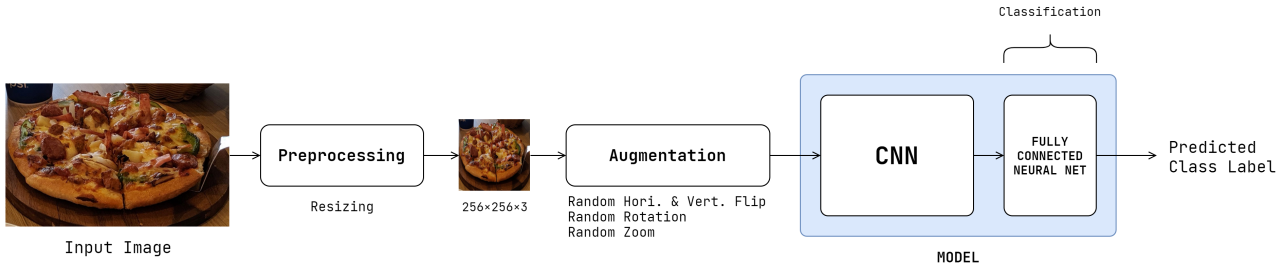


Fig. 8: The high-level workflow of how an image is being fed into the model in order to get the output class label.

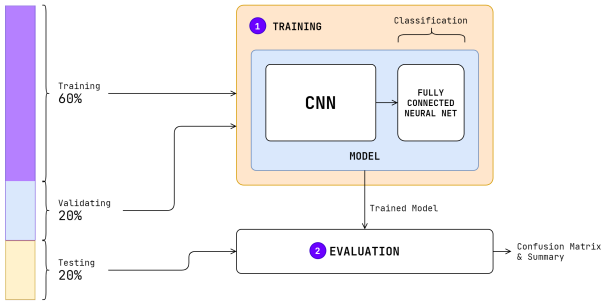


Fig. 9: The workflow that is used when training a model from scratch.

commerce during training. After all training data or an epoch underwent training, the model will be validated with the validation data in terms of its loss and accuracy to indicate whether the model overfit or underfit the training data or not. After certain stopping criteria either through early stopping or a number of epochs reached, the training process will stop.

2. **Evaluation Phase** - After the training phase has completed, the trained model will be fed with another set of data called test data. This is to validate the performance of the model to see how well it can perform on unseen data. In this phase, the predictions and the ground truths of the test data will be used to evaluate the performance via a confusion matrix. With a confusion matrix, evaluation metrics such as accuracy, precision, recall, and F1 score could be calculated.

F Custom-made models

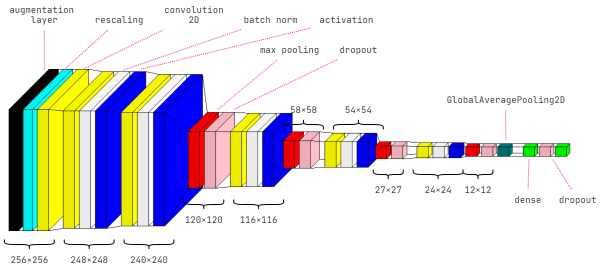


Fig. 10: Architecture of the "Custom 1" model.

In this study, we have built 4 custom models for the Thai food image classification task. This is because we desired to explore configuration variations of CNN that perform the best given a limited amount of time and computing resources. From figure 10, 11, 12 and 13, the

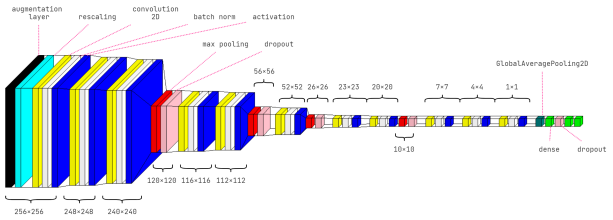


Fig. 11: Architecture of the "Custom 2" model.

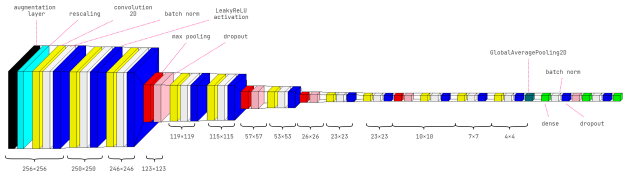


Fig. 12: Architecture of the "Custom 3" model.

models are denoted as blocks. Please note each different type of layer is uniquely color-coded. All models are different in the sense that one has more parameters to adjust than the other. Besides, for the `custom3` and `custom4`, the LeakyReLU is the activation function of choice, however, `custom1` and `custom2` use the ReLU activation function instead.

G Transfer Learning method

On the other hand, we also choose to experiment with the transfer learning method to evaluate the effectiveness of popular pre-trained models on our Thai food domain application which only has a small set of data. To integrate the model for transfer learning, we must get a pre-trained model and remove its original fully connected layers because it cannot be applied to our domain. The aforementioned fully connected layer must be replaced with another one with randomly initialized weights. For transfer learning, there are 3 main phases:

1. **Training Phase** - Similar to the one in the previous method, after the model is defined, the model is fed with the training data in order to train its re-

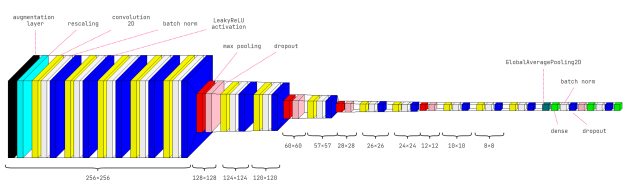


Fig. 13: Architecture of the "Custom 3" model.

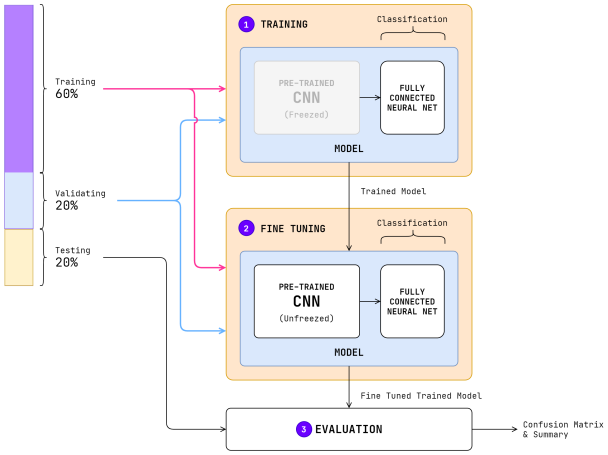


Fig. 14: The workflow that is used when training a model from a pre-trained model via transfer learning.

cently replaced dense layers. Forward propagation will happen normally while backward propagation will only update the weights of the aforementioned dense layers. The stopping criteria of training are identical to the previous method. This phase essentially makes sure that newly added dense layers are compatible with the output of the pre-trained model that is dictated by input images.

2. **Fine Tuning Phase** - After the training process is converged, the model is ready for a slight fine-tuning in its pre-train models. In this phase, the pre-trained model weights will be unfrozen allowing it to be slightly changed according to the optimizer. In this phase, forward and backward propagation takes place using the same training data. This phase only lasts only a handful of epochs to prevent overfitting.

3. **Evaluation Phase** - This phase is identical to the one previously mentioned in the training from scratch approach.

H Models from transfer learning

This section lists the pre-trained models that we used for transfer learning.

1. **DenseNet** [13] - This model has developed by Gao Huang. The Dense Convolutional Network connects each layer to every other layer in a feed-forward process. Their network has $\frac{L(L+1)}{2}$ direct connections. Due to the idea from recent works that convolutional networks can be substantially

deeper, more accurate, and efficient if they hold shorter connections among layers close to the input and the output.

2. **ResNet101v2** [14] - This model has developed by Kaiming He *et. al.* They analyze the propagation formulations behind the residual building blocks. From the study, they encounter problems and recommend using identity mappings as the skip connections and after-addition activation. This motivates them to propose a new residual unit, which makes training easier and improves generalization.

3. **ResNet50v2** [14] - This model has developed by Kaiming He *et. al* due to they want to do deeper neural network training. They reformulate the layers as learning residual functions concerning the layer inputs. The result proves that these residual networks are easier to optimize, and can reach higher accuracy from increased depth.

4. **Xception** [15] - This model has developed by Chollet *et. al.* The model uses inception modules in convolutional neural networks with the depth-wise separable convolution operation. They need to do a novel deep convolutional neural network architecture. They inspired by Inception which has been replaced with depth-wise separable convolutions.

I Mitigating Underfitting & Overfitting

Since both underfitting and overfitting characteristics or so-called the bias-variance trade-off tremendously affect the final performance of our models, we would like to expand on what are the strategies we used to avoid both when fitting those models.

In the early iteration of our custom model design, there were quite a several instances where the model cannot properly learn from the data. Over a few iterations, training losses does not decrease but rather fluctuate with no sign of decreasing. One of the first thing that we investigate is that we try making the model more complex in the sense that we add more layers, kernels or units to make the model has sufficient parameters to optimize upon. However, if making the model more complicated does not work, we choose to investigate the learning rate. There was an instance where decreasing the learning rate as small as 10^{-5} could make the model learn.

On the other hand, despite frequent underfitting occurrences, for some models especially transfer learning ones, overfitting is very common with CNN and needed to be resolved to make certain that the model is generalized with unseen data. One of the many things that we extensively tried is to introduce dropout. Dropout is a technique that randomly freezes parameters to make it less susceptible to too much learning that causes overfit. Moreover, sometimes we also introduce L2 regularization to some of the kernel or weight update. Early stopping is also used frequently to avoid parameters being too fitted for the data. Furthermore, as the last resort, we also reduce the complexity of the model itself to reduce the number of parameters that can be adjusted. This is to make sure that too many adjustments will not lead to overfitting with the training data.

J Batch Normalization

To make the optimization of parameters easier to converge, we decided to incorporate batch normalization into our custom models [16]. Since during training all parameters including weights and kernels are continuously updated throughout the process, optimizing parameters of a hidden layer when the preceding layer also gets its parameters updated is proved to be troublesome. For example, the weights of the current layer are updated with an assertion that the previous layer outputs values with a given distribution; however, in reality, such assertion does not hold. This forces the optimizer to use a much lower learning rate. With batch normalization in place, we can use a much larger learning rate to speed up the convergence of the training process. Moreover, we also gain benefits from its slight regularization effects making models less overfit.

K Optimizer

As for the optimizer, instead of using the original mini-batch gradient descent, we decide to adopt Adam (Adaptive Moment Estimation) optimizer [17] with the default configuration from Keras. This is because it offers an adaptive learning rate, which is beneficial for training speed. Adam achieves such by using a large learning rate initially and subsequently lowering it down as the training progresses to make sure that the parameters will not oscillate when approaching the global minimum of the loss function. Moreover, for our custom models, we decided to opt for the Adam optimizer with running average to ensure that the loss is not fluctuate when a continuous stream of augmented images flow into the model when training.

IV IMPLEMENTATION

With the already existing computing resources offered by the faculty, we selected TensorFlow with Keras as

a framework to implement our experimentations. We used Python in an Anaconda environment with Jupyter notebooks to facilitate our workflow.

A Dataset Loading

As mentioned earlier, we aimed to segment the dataset into 3 portions with stratified sampling; however, as far as we investigated, there is no Keras-TensorFlow native way to perform such sampling. Therefore, our workaround is to rely on scikit-learn's `train_test_split` method. Subsequently, the dataset, which is a folder that has 38 subfolders each of which represents each class, is read file-by-file as an URL string. Moreover, the class label of each image is inferred from the URL. With both the list of image file URLs and the list of labels, the dataset can be partitioned into 3 parts with stratified sampling. After that, each portion is transformed into TensorFlow's tensor to make it performant for the subsequent training pipeline using a custom-implemented image loader method which also includes an image resizing process.

B Dataset Loading

Since TensorFlow allows the image augmentation and rescaling process is implemented as a layer of a model, we leverage that characteristic to improve the efficiency of the training process. Please note that the data augmentation layer will not be executed during the time of evaluation and testing. This is one of the efforts that we used to combat sluggish model training caused by our complex features in food images. Moreover, the same configuration is also done for the pixel value rescaling.

C Model Training

After a model is defined with layers, activation functions, normalization, and regularization, the training process of each model is carried out on an NVIDIA Geforce RTX 2080ti as provided by the faculty for the senior project. The number of epochs and the batch size is varied depends on each model. To be specific, the number of epochs is selected based on how losses of the model over epochs are. For example, if the losses fluctuate frequently, the larger batch size is used to ensure each gradient update takes more images into an account to ensure the stability of model losses.

V EXPERIMENTAL RESULTS

In the experiment, as mentioned previously, we trained our models on a faculty-provided computer. It is equipped with Intel Core i9-9900X with 128GB of memory, and 2 NVIDIA GeForce RTX 2080ti. At a time, the machine can be used to train up to 3 models simultaneously using CPU and all GPU.

We conducted a comparative experiment for six classification models trained from our dataset. As mentioned earlier, the dataset consists of 38 classes each of which has 250 images. Totally there are 9,500 images. We partitioned the dataset into 3 parts: training, validation, and testing. The experiment results as shown in table 2 are evaluated from the predictions made by the models on the testing set.

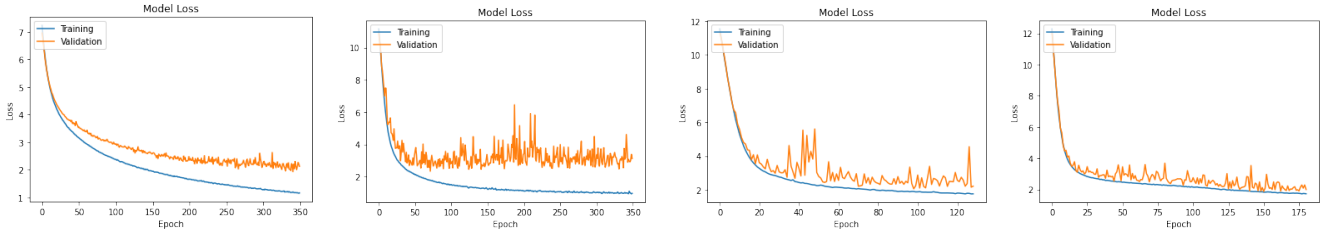
A Custom models

Although all models' hyperparameters are manually fine-tuned to some extent, it can be observed from the table that custom models perform very poorly on the testing data even when `custom1` and `custom2` are set to converge after 350 epochs. By observing from the loss over the iteration plot in figure 15, the first 2 models clearly overfitted and underfitting the training data considering its poor performance, and a relatively large gap in between training and testing loss, whereas the other 2 do not overfit but rather underfit both training and testing.

For the first two models, we hypothesized that regularization hyperparameters are not sufficiently intense even with batch normalization. However, for `custom3` and `custom4`, the amount of regularization was increased. Moreover, the reason why all are underfitting is that they are not complex enough to learn the complicated features of food images of a vast diversity of light conditions, unlike pre-trained models that have way more complexity.

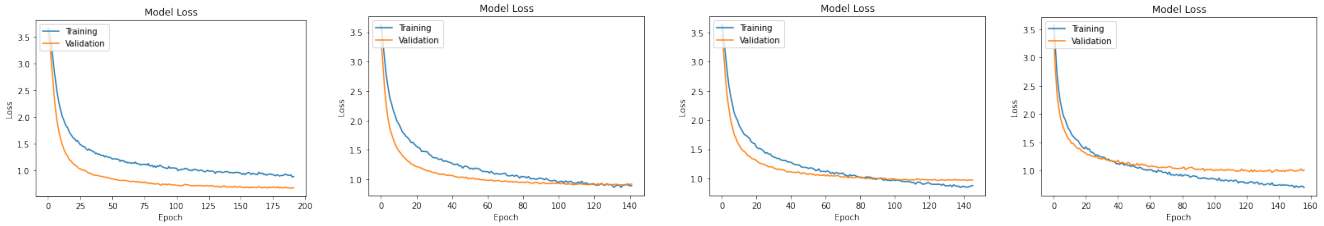
On top of that, another problem that we noticed in all custom models is that the validation loss plot (figure 15) shows intense oscillations, especially for the `custom1` and `custom2`. We hypothesized that this is because models' inability to generalize on augmented data is showing its effects as well as the small size of a validation batch. The fixated or unscheduled learning rate can also be the culprit of the oscillations, but this may be unlikely because they are utilizing Adam optimizer. Nevertheless, the last two models' loss plots in figure 15c, 15d show that the oscillation is lessened when comparing to the first two models.

Despite the obstacles, according to the reported F1 scores in table 2, the first custom model performed worse than the second one, because the model is unable to generalize itself to unseen data due to lack of model complexity. Overall, custom models took tremendously long to minimize their loss because their initial loss is very



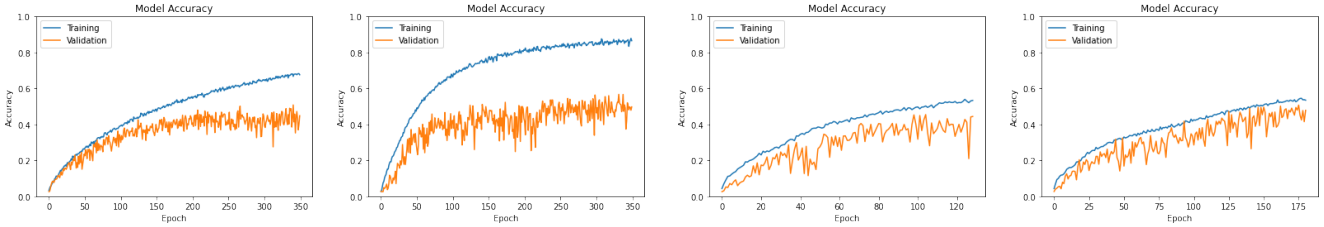
(a) *Custom 1* model **(b)** *Custom 2* model **(c)** *Custom 3* model **(d)** *Custom 4* model

Fig. 15: The training loss and validation loss over epochs during the training process of custom models.



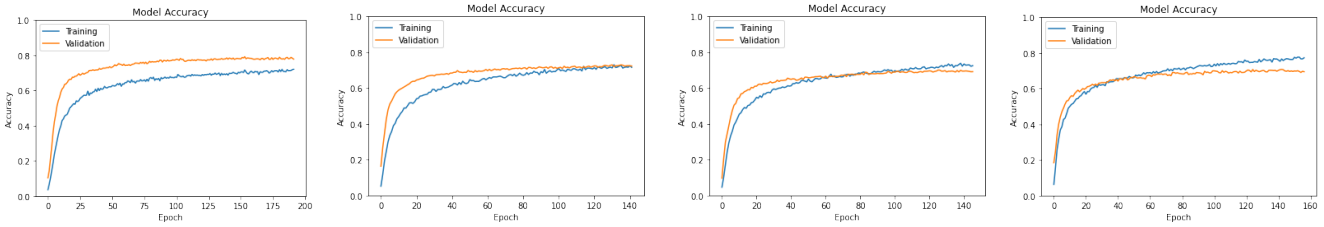
(a) *DenseNet201* model **(b)** *ResNet101v2* model **(c)** *ResNet50v2* model **(d)** *Xception* model

Fig. 16: The training loss and validation loss over epochs during the training process of transfer learning models.



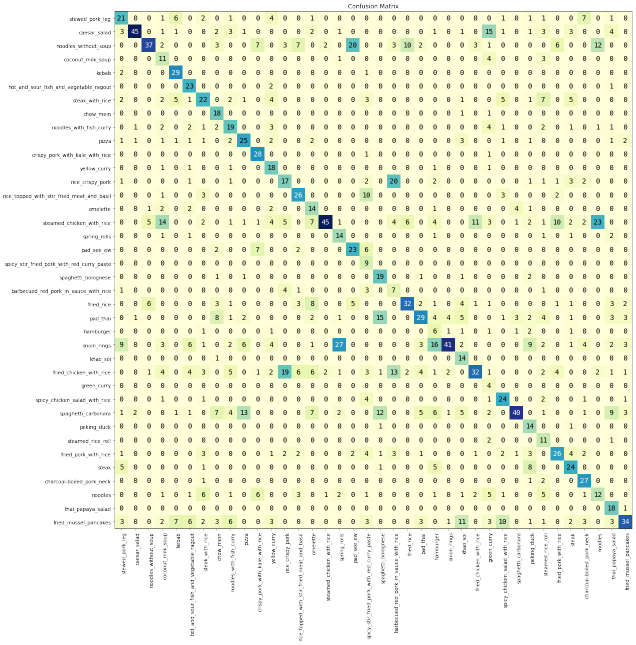
(a) *Custom 1* model **(b)** *Custom 2* model **(c)** *Custom 3* model **(d)** *Custom 4* model

Fig. 17: The training accuracy and validation accuracy over epochs during the training process of custom models.

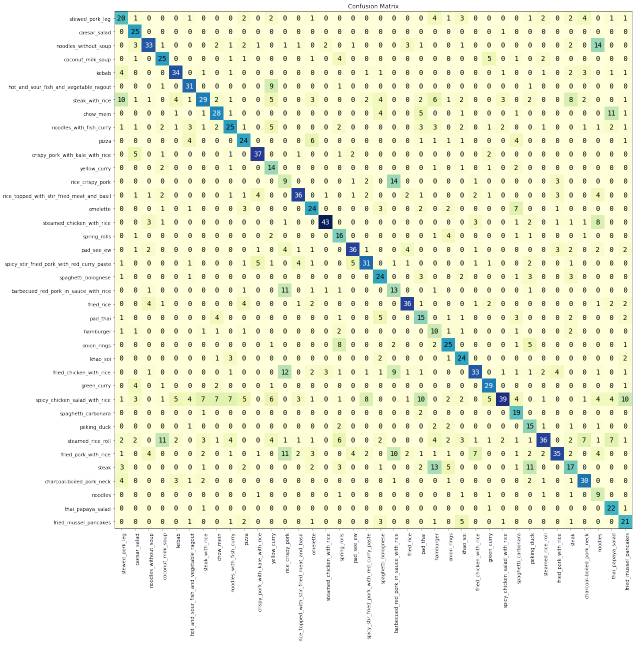


(a) *DenseNet201* model **(b)** *ResNet101v2* model **(c)** *ResNet50v2* model **(d)** *Xception* model

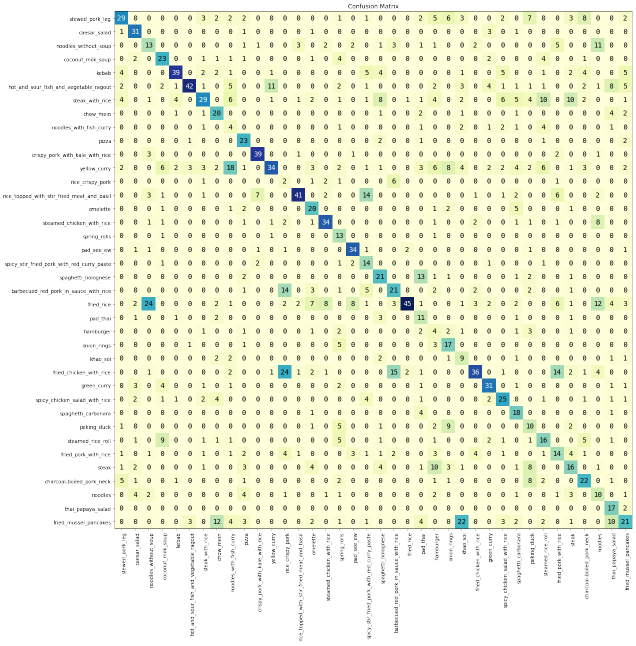
Fig. 18: The training accuracy and validation accuracy over epochs during the training process of transfer learning models.



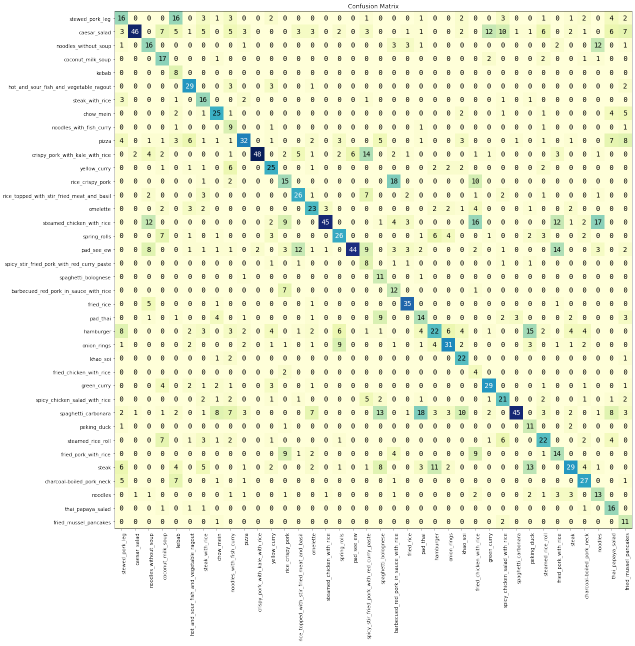
(a) *Custom 1* model



(b) *Custom 2* model

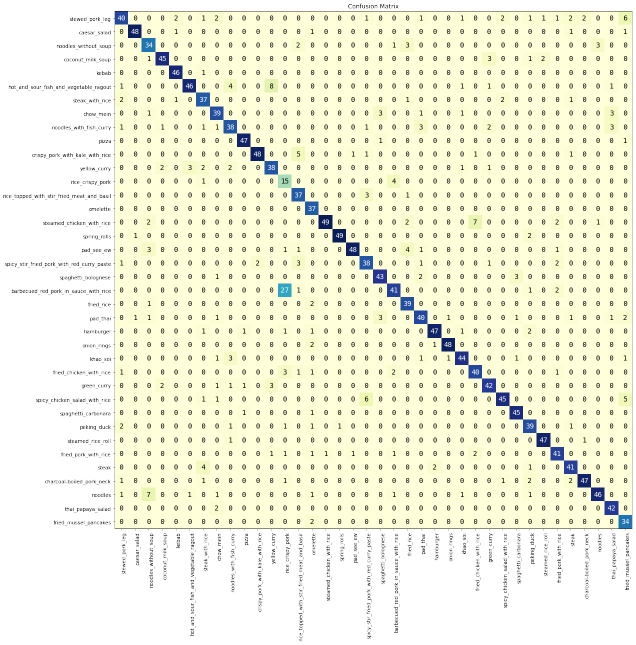


(c) *Custom 3* model

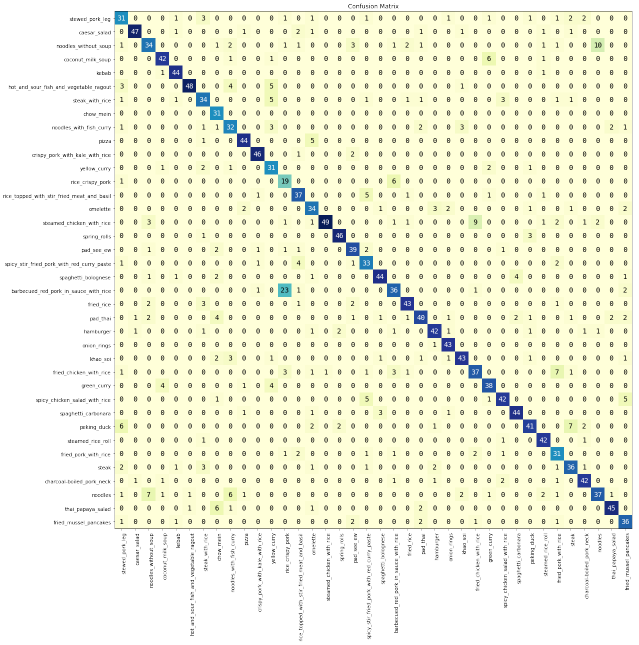


(d) *Custom 4* model

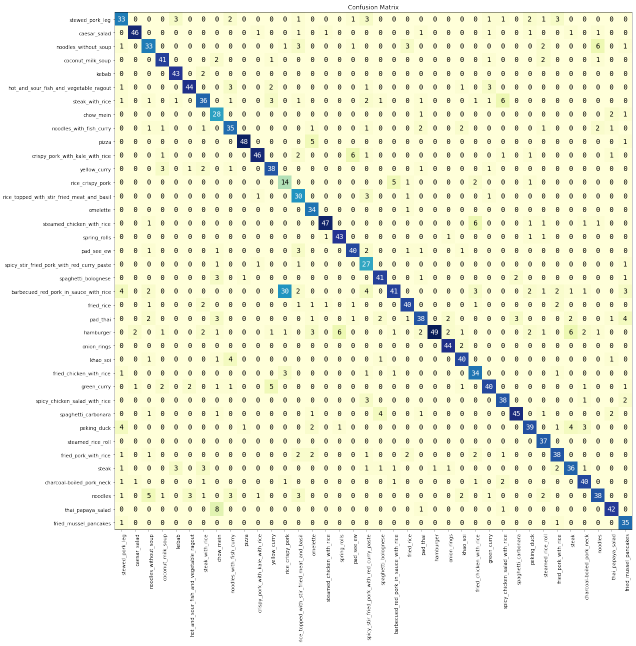
Fig. 19: The confusion matrices of the custom models' predictions on the test data.



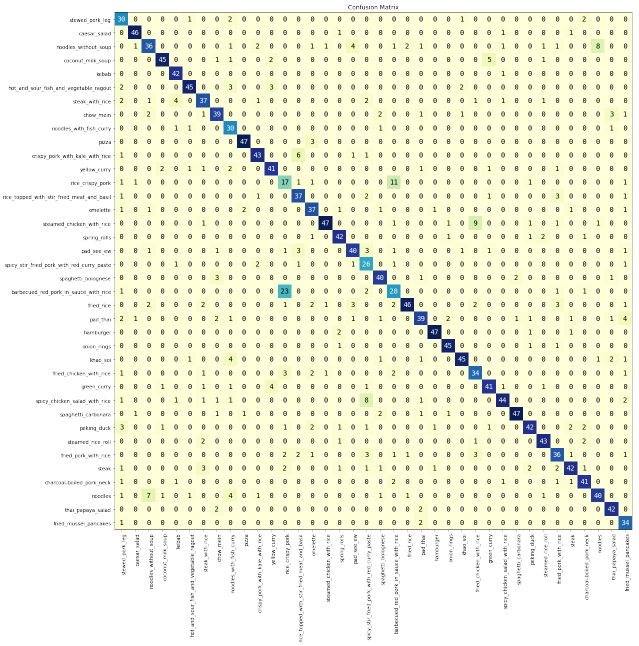
(a) *DenseNet201* model



(b) *ResNet101v2* model



(c) *ResNet50v2* model



(d) *Xception* model

Fig. 20: The confusion matrices of the transfer learning models' predictions on the test data.

Table 2: The prediction results of each model on the test data

Model name	n_b	α	Accuracy	Precision	Recall	F1 Score	$t_{training}$	$t_{finetuning}$
Custom 1	32	10^{-5}	0.45	0.45	0.59	0.45	14 hours	-
Custom 2	32	10^{-4}	0.51	0.51	0.56	0.51	16 hours	-
Custom 3	32	10^{-4}	0.45	0.45	0.49	0.43	4 hours	-
Custom 4	20	10^{-4}	0.45	0.45	0.55	0.44	17 hours	-
DenseNet201	24	10^{-4}	0.84	0.84	0.84	0.83	67.2 minutes	16.25 minutes
ResNet101v2	24	10^{-4}	0.78	0.78	0.79	0.78	47 minutes	14 minutes
ResNet50v2	24	10^{-4}	0.77	0.77	0.79	0.77	34 minutes	9 minutes
Xception	24	10^{-4}	0.79	0.79	0.80	0.79	45 minutes	17 minutes

Note: n_b denotes the batch size, α denotes the learning rate used for training. $t_{training}$ denotes training duration. $t_{finetuning}$ denotes fine-tuning duration (for transfer learning).

high due to randomly initialized parameters, however, both `custom3` and `custom4` use early stopping; thus, the training time is significantly minimal, but yielding a similar performance to `custom1` and `custom2`. Figure 19 shows the confusion matrix of each custom model.

Furthermore, we strongly believe there are performance implications caused by the dataset that we used and how each image are resized. Concretely, when an image is put into the dataset, image size and image aspect ratio is not fixated; therefore, when during data loading and preprocessing, some images that are wide might get naively resized to a square. This might somehow make the model underfit the training data.

B Models from transfer learning

For the transfer learning models, table 2 shows that pre-trained models used for transfer learning accelerated the training process significantly due to fewer parameters to be optimized in the training phase while bringing the evaluation performance as far as 0.83 for F1-score on the test data. Moreover, the loss plots, figure 18, show that these models are neither overfitting nor underfitting food image data. These extraordinary performances are not surprising considering that these pre-trained models are comprised of complex layers and trained using the ImageNet dataset, which has about 14 million images. Therefore, it is certain that these pre-trained models can extract important general features of images such as edge, shape, color, etc. As a result, they can be adapted to other applications such as Thai food recognition without any hassle. Figure 20 shows the confusion matrix of each custom model.

According to the confusion matrices of the transfer learning models, while there are mostly accurate predictions, there are also misclassifications that are caused by how similar those food classes are. For instance, it can be observed that most classifiers mispredict certain classes of foods as illustrated in the confusion matrices of transfer learning models (figure 20) as bold non-diagonal squares. This is maybe caused by the fact that there are certain kinds of foods that are visually similar. For instance, `barbecued_red_pork_in_sauce_with_rice` (known as ข้าวหมูแดง) and `rice_crispy_pork` (known as ข้าวหมูกรอบ) are usually misclassified by our models because the composition of such menu would usually include rice, boil eggs, sauce, and either crispy pork or barbecued red pork which is the only difference between both classes. Another example could be the `noodle` class and the `noodle_without_soup` class. The `steamed_chicken_with_rice` class (known as ข้าวมันไก่) and `fried_chicken_with_rice` (known as ข้าวมันไก่ทอด) is another pair of such misclassification.

VI CONCLUSION

In this work, we conducted a comparative study of multiple deep learning-based models. We compared custom-made models and models from transfer learning for the Thai food recognition task. To do so, we also created a new dataset that is consisted of 9,500 images from over 38 food types. The models were built using Keras with TensorFlow. After experimentation and fine-tuning, we discovered that custom-made models need tremendous efforts to make them viable for general usage such that there is no underfitting and overfitting problem. Furthermore, we found that hyperparameter selection is very crucial for one to achieve acceptable classification performance. On the other hand, the transfer learning models are much faster and generally yield better results because it utilizes feature extraction capability from one domain to another. The final results

show that transfer learning-based models are lightyears ahead of our custom-made models in terms of classification performance.

In future work, we will further improve classification results by introducing food segmentation and food detection in order to mitigate the assumption that each food image can only contain one food item. Moreover, we will also utilize more complex classification models to improve performance. Furthermore, methodical hyperparameter fine-tuning is also another regime that we would like to explore.

References

[1] Parisa Pouladzadeh, Gregorio Villalobos, Rana Almaghrabi, and Shervin Shirmohammadi. A novel svm based food recognition method for calorie measurement applications. pages 495–498, 07 2012.

[2] Yoshiyuki Kawano and Keiji Yanai. Real-time mobile food recognition system. pages 1–7, 06 2013.

[3] Parisa Pouladzadeh, Shervin Shirmohammadi, and Rana Almaghrabi. Measuring calorie and nutrition from food image. *Instrumentation and Measurement, IEEE Transactions on*, 63:1947–1956, 08 2014.

[4] Kiyoharu Aizawa, Yuto Maruyama, He Li, and Chamin Morikawa. Food balance estimation by using personal dietary tendencies in a multimedia food log. *IEEE Transactions on Multimedia*, 15:2176, 12 2013.

[5] Hokuto Kagaya, Kiyoharu Aizawa, and Makoto Ogawa. Food detection and recognition using convolutional neural network. 11 2014.

[6] Sinem Aslan, Gianluigi Ciocca, Davide Mazzini, and Raimondo Schettini. Benchmarking algorithms for food localization and semantic segmentation. *International Journal of Machine Learning and Cybernetics*, 06 2020.

[7] Parisa Pouladzadeh and Shervin Shirmohammadi. Mobile multi-food recognition using deep learning. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 13:1–21, 08 2017.

[8] Chakkrit Termritthikun, Paisarn Muneesawang, and Surachet Kanprachar. Nu-innet: Thai food image recognition using convolutional neural networks on smartphone. *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, 9:2289–8131, 08 2017.

[9] Gianluigi Ciocca, Paolo Napoletano, and Raimondo Schettini. Food recognition: A new dataset, experiments, and results. *IEEE Journal of Biomedical and Health Informatics*, PP:1–1, 12 2016.

[10] Xi-Jin Zhang, Yi-Fan Lu, and Song-Hai Zhang. Multi-task learning for food identification and analysis with deep convolutional neural networks. *Journal of Computer Science and Technology*, 31:489–500, 05 2016.

[11] Eduardo Aguilar, Marc Bolaños, and Petia Radeva. Regularized uncertainty-based multi-task learning model for food analysis. *Journal of Visual Communication and Image Representation*, 60:360 – 370, 2019.

[12] Ukrit Tiankaew, Peerapon Chunpongthong, and Vacharapat Mettanant. A food photography app with image recognition for thai food. pages 1–6, 07 2018.

[13] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks, 2018.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks, 2016.

[15] François Chollet. Xception: Deep learning with depthwise separable convolutions, 2017.

[16] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[17] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.