T.C.
BAHÇEŞEHİR ÜNİVERSİTESİ

IMPROVING CONTEXT EMPOWERED DEEP NEURAL NETWORKS

Master Thesis

CAN KURT

ISTANBUL, 2021

T.C.
BAHÇEŞEHİR ÜNİVERSİTESİ

GRADUATE SCHOOL
COMPUTER ENGINEERİNG MASTER'S PROGRAM

IMPROVING CONTEXT EMPOWERED DEEP NEURAL NETWORKS

Master Thesis

CAN KURT

Supervisor: ASST. PROF. ÖVGÜ ÖZTÜRK ERGÜN

ISTANBUL, 2021

T.C.

BAHCESEHIR UNIVERSITY

GRADUATE SCHOOL

26/06/2021

MASTER THESIS APPROVAL FORM

Program Name:	Computer Engineering Master's Program
Student's Name and Surname:	Can Kurt
Name of The Thesis:	IMPROVING CONTEXT EMPOWERED DEEP NEURAL NETWORKS
Thesis Defense Date:	26/06/2021

This thesis has been approved by the Graduate School which has fulfilled the necessary conditions as Master thesis.

Assoc. Prof. Dr. Burak KÜNTAY

Institute Director

This thesis was read by us, quality and content as a Master's thesis has been seen and accepted as sufficient.

	Title/Name	**Signature**
Thesis Advisor's	Asst. Prof. Övgü Öztürk Ergun	
Member's	Asst. Prof. Tarkan Aydın	
Member's	Asst. Prof. Mustafa Özuysal	

ACKNOWLEDGMENTS

This thesis is dedicated to my respectable family.

I would like to express my gratitude to my supervisor Asst. Prof. Övgü ÖZTÜRK ERGÜN for encouraging and challenging me throughout my thesis studies.

I would also like to thank my friend Bekir Furkan Kesgin for all of his help on collecting our datasets.

ISTANBUL, 2021 CAN KURT

ABSTRACT

IMPROVING CONTEXT EMPOWERED DEEP NEURAL NETWORKS

Can Kurt

Computer Engineering Master's Program

Supervisor: Assist. Prof. Övgü Öztürk Ergün

May 2021, 45 Pages

In this study context empowered Deep Neural Networks are tried to be Improved with the fusion. Context empowered Deep Neural Networks can be regarded as the first part of this study and they examined in the Celik (2021) in detail. This study focuses on using the context information that is generated by a Deep Neural Network, to improve an existing Deep Neural Network structure with the fusion to reach a better accuracy. Firstly, a Turkish cuisine food dataset is collected, Turkish food consists of many ingredients and it is hard to classify with regular methods known on the literature. Multiple sized datasets are collected and some cut to create a texture dataset to be tested. Furthermore, a web application is created to collect more food images that can use existing trained models to predict food from image, that way it is ensured that users can test our models while helping us collecting more images. Before testing the results with fusion several Deep Neural Network structures tested with multiple sizes of Turkish cuisine food datasets to create a baseline. After that a texture test is conducted to determine the best resolution for first fusion test. Finally, different types of fusion tests are done with the collected datasets to test the improvement.

Keywords: Deep Neural Network, Convolution Network, Fusion, Context Information, Web Application.

ÖZET

BAĞLAM BİLGİSİ İLE GÜÇLENDİRİLMİŞ DERİN SİNİR AĞLARININ GELİŞTİRİLMESİ

Can Kurt

Bilgisayar Mühendisliği Yüksek Lisans Programı

Tez Danışmanı: Dr. Öğr. Üyesi Övgü Öztürk Ergün

Mayıs 2021, 45 Sayfa

Bu çalışmada güçlendirilmiş Derin Sinir Ağları füzyon ile geliştirilmeye çalışılmıştır. Bağlamla güçlendirilmiş Derin Sinir Ağları, bu çalışmanın ilk bölümü olarak kabul edilebilir ve Celik (2021) çalışmasında ayrıntılı olarak incelenebilir. Bu çalışma, daha iyi bir doğruluğa ulaşmak için mevcut bir Derin Sinir Ağı yapısını füzyonla iyileştirmek için başka bir Derin Sinir Ağı tarafından üretilen bağlam bilgisini kullanmaya odaklanmaktadır. İlk olarak, bir Türk mutfağı yemek veri seti toplanmıştır, Türk yemekleri birçok bileşenden oluşmaktadır ve literatürde bilinen yöntemlerle sınıflandırılması zordur. Birden çok farklı boyutlarda veri kümesi toplanmıştır ve bazıları kırpılarak test edilemk üzere bir doku veri kümesi de oluşturmuştur. Ayrıca, yiyecekleri görüntüden tahmin etmek için halihazırda eğitimlmiş modelleri kullanabilen daha fazla yiyecek görüntüsü toplamak için bir web uygulaması oluşturulmuştur, böylece kullanıcıların daha fazla görüntü toplamamıza yardımcı olurken modellerimizi test etmeleri sağlanmıştır. Sonuçları füzyonla test etmeden önce, birkaç Derin Sinir Ağı yapısı, bir alt sınır oluşturmak için çeşitli boyutlarda Türk mutfağı gıda veri kümeleri ile test edildimiştir. Bundan sonra, ilk füzyon testi için en iyi çözünürlüğü belirlemek için bir doku testi yapılırmıştır. Son olarak, iyileştirmeyi test etmek için toplanan veri kümeleriyle farklı türlerde füzyon testleri yapılmıştır.

Anahtar Kelime: Derin Sinir Ağı, Evrişim Ağı, Füzyon, Bağlam Bilgisi, Web Uygulaması.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

API	:	Application Programming Interface
CNN	:	Convolutional Neural Network
DNN	:	Deep Neural Network
SGD	:	Stochastic Gradient Descent
YOLO	:	You Only Look Once

1. INTRODUCTION

After the success of the AlexNet Krizhevsky (2012) on the ImageNet competition in 2012, the interest for deep neural networks increased. Even though the idea of artificial neural networks dates back to the 1950s, it wasn't a viable option until now. Today deep learning applications use GPUs that are more accessible. With this computational barrier is breached, deep learning started to become more usable. But for deep learning to work, there was one more piece to the puzzle, and it was the data. After today's massive usage of the internet, all types of data started to be available for everybody. With these improvements, deep learning started to show its capabilities on various tasks.

The aim of this project is to improve the context-empowered deep neural networks with various methods. In a previous study Celik (2021), extracting contextual information from food images is examined. This contextual information has to be added to the DNNs to improve the accuracy. In this project effects of the "fusion" method are tested, it is used to add the context information to the DNNs. Fusion is tested in various ways to determine the best usage of the context information.

This study is a part of an ongoing project, because of that there was already a dataset that has been collected. During this project data collection continued and dataset size is increased. To collect data more efficiently a detailed full-stack web application is created as a part of this project.

Before the fusion tests, many different well-known CNN architectures were examined for testing. Such as DenseNets Huang (2016) and EfficientNets Tan (2019). After some preliminary tests, more in-depth tests are also done with these models. These tests are conducted to determine the best resulting CNN models, those models are later used to be improved with fusion and context information.

After regular models are tested, a resolution test is done. This test aims to determine the effects of the resolution during training on very small images. This test is required to find

the best resolution for ingredient images to train. Since ingredient images are cropped from regular food images, they are very small.

After resolution tests done for the ingredient images, a fusion test is conducted. In this test, fusion is used for adding ingredient images to DNNs. In these tests, an ingredient image is extracted from a food image, and both the food image and the ingredient image are fed into the DNN. It is expected that the network will learn the ingredient occurrences on the food images, and after the addition of this context information results are expected to go higher.

Fusion is also used with context information that is created separately from the DNN. This information is added to the DNN as an ingredient vector. It is first tested with an ingredient vector that is created by hand for the dataset.

Finally, after these tests instead of using hand-crafted ingredient vector already extracted vectors used from the project Celik (2021).

2. GENERAL DEEP NEURAL NETWORK STRUCTURE

Deep neural networks (DNNs) are a sub-category of artificial neural networks (ANNs). ANNs are inspired by the functions and the structures of the cells in the brain. The most distinguishable feature of a DNN is the hidden layers. DNNs have multiple hidden layers, the "deep" in the name comes from this multi-hidden layered structure. Other than having multiple hidden layers DNNs have all the other characteristics of artificial neural networks. DNNs are useful because they can create "connections" by extracting information from large datasets which is hard or impossible for humans to see. The process of making these connections is called "training", and a "trained" DNN can use its connections to make predictions, and by doing that it can solve real-life problems.

2.1 CONVOLUTIONAL NEURAL NETWORK

CNNs are a sub-category of DNNs and they are mostly used on image classification, localization, or detection tasks. Before CNNs there were no real-time detection or classification algorithm that is general enough to be used on many different objects. CNNs revolutionized computer vision. A CNN is consisting of 2 main parts, a feature extraction part and a regular DNN based fully connected decision-making part. In the feature extraction part, different types of filters are applied to the images to extract the most information from the image, this extracted information is stored in feature maps. Convolutional layers can be followed by pooling layers for decreasing the size of the obtained feature maps during the convolution. Multiple convolutional layers can be applied for more feature extraction. After every convolutional layer the filters get more and more complex and as a result of that CNN starts to extract more and more complex feature maps to detect more complex objects. After the feature extraction step, a set of fully connected layers gets the flattened information from the convolutional layers and makes the decision that is appropriate to the task that CNN is used.

2.1.1 DenseNet

Recent DNN architectures show that DNNs can be significantly deeper, but as a result of that a lot more parameters are needed to be used on those DNNs. DenseNet Huang (2016) tries another way, instead of a very deep network DenseNet is a very dense network as the name suggests. A regular CNNs convolutional layer can only pass the extracted information to the next layer, DenseNet can do more. DenseNet has a layer called “Dense Layer” or “Dense Block” this layer is a collection of different layers but in the base of it there are multiple convolutional layers, and they not only pass the extracted information to the next layer but also to every other layer after itself. With the dense layers, DenseNet’s layers can learn a “collective knowledge”. This way of using dense layers instead of deeper networks uses fewer parameters and they also alleviate the vanishing gradient problem. DenseNet has many versions and it is used in many different areas of computer vision. Some examples of those are; classification, semantic segmentation, quantization, and object detection.

Figure 2.1: Regular DNN structure

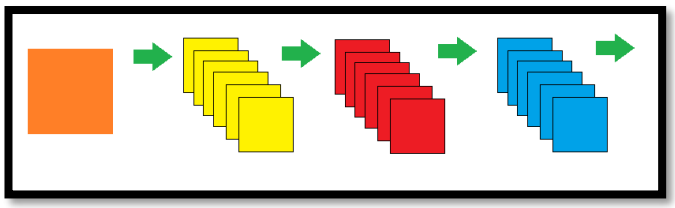
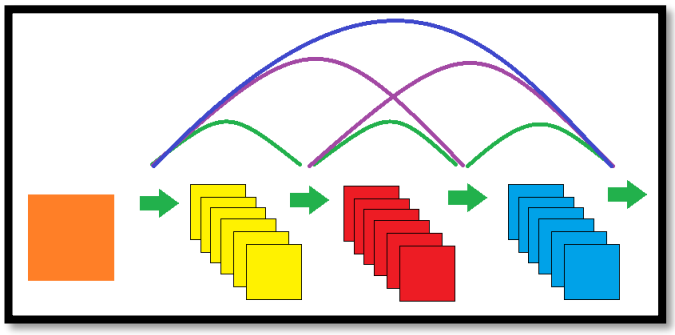


Figure 2.2: DenseNet structure



2.1.2 YOLO

YOLO, "You Only Look Once" Redmon (2015), is a real-time object detection algorithm that uses a CNN as a backbone. The biggest difference between YOLO and older object detection algorithms is that YOLO is a one-step detector. A One-step detector means that all the detection process is a result of a single DNN result. Older methods use a two-step detection method. First, a proposed area for the possible object is found then a CNN finds the class of the object. Since two-step detection is a slow process YOLO with its one-step detection is stands out as a real-time algorithm for object detection.

New YOLO versions added more functionality, stability, and significant speed increases. Such as anchor boxes that were added on YOLOv2 Redmon (2016) for fixing YOLO's inability to find multiple objects on crowded images or mosaic augmentation method that was added on YOLOv4 for better training.

Original YOLO authors created the Darknet, an open-sourced deep learning framework. The most popular algorithm of the Darknet is YOLO but there are more CNN models on the framework. Today a fork of this framework is continued after the original author of the framework decided to discontinue the project. The author's explanation for discontinuation is "*I stopped doing CV research because I saw the impact my work was having. I loved the work but the military applications and privacy concerns eventually became impossible to ignore."* YOLOv4 Bochkovskiy (2020) is published by the author of the new Darknet repository.

Because of its many successes, YOLO is decided to be used on the classification tests with our food datasets on this project.

2.1.3 EfficientNet

EfficientNet Tan (2019) is not only a convolutional neural network architecture it is also a scaling method. The idea behind the EfficentNet is systematically and uniformly scaling the network features depth, width, and resolution. EfficientNet manages to get better

accuracy by carefully scaling the network, decreasing the network size also increases the network speed. In their paper, it is shown that EfficentNet-B7 their newest model, achieves a state-of-the-art ImageNet top-1 accuracy of 84.4 percent and top-5 accuracy of 97.1 percent. These results make EfficientNet a great candidate for testing it on our food data on this project.

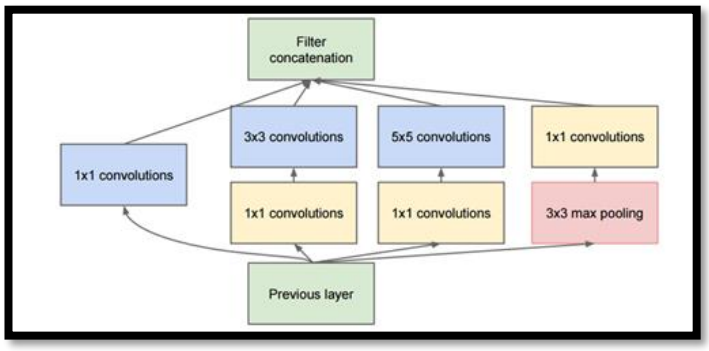
2.1.4 ResNet

The idea behind ResNet He (2015) comes from “pyramidal cells” that are inside the “cerebral cortex”, a residual neural network is a DNN family that utilizes jumping over some layers or skipping connections for solving vanishing gradient problem. This problem is a common problem on very deep DNNs. There are many different ResNet versions and they are used for different tasks. Such as; classification, detection segmentation, and Self-Supervised Learning. Since it is a working and well-known architecture it is used in this project.

2.1.5 Inception

Inception Szegedy (2014) networks are designed by Google to decrease the amount of computational power needed to train the deeper networks. They allow deeper networks to be used with more moderate hardware. They use “inception blocks” to achieve this. Inception blocks have a set of convolution operations with different windows sizes so the network itself can decide which convolution will be more useful. Also, there are clever techniques used in inception layers like bottlenecks to decrease computational cost even more.

Figure 2.3: Inception block



2.1.6 VGG

When the AlexNet Krizhevsky (2012) came out in 2012 it was the best CNN architecture for image classification at the time, and it stayed in this position until VGG Simonyan (2015) came out in 2015. VGG uses smaller window sizes on its convolutional layers. VGG also uses the ReLU activation function on all of its hidden layers, this cuts the training time significantly when compared to the AlexNet. VGG has 3 fully connected layers at the end of the network, the first two layers have 4096 units and the last one has 1000. VGG is a simple network and it is still usable on simple classification or detection tasks.

2.2 SCORE FUNCTIONS

2.2.1 SGD

The gradient descent is an algorithm that is first proposed in the 1950s, SGD is a variant of it. Today SDG is the most common method that is used to update each parameter of a DNN. The difference between regular gradient descent and SGD is instead of performing computations on the entire dataset, SGD only computes on a small chunk of the dataset that is randomly selected. Performance-wise SGD is mostly capable of producing the same performance as the regular gradient descent especially when the learning rate is low. Since it is one of the most useful optimizers SGD will be used during the tests.

2.2.2 Adam

Adam is a gradient-based optimization algorithm. It combines all the advantages from known algorithms like SGD, RMSProp, and AdaGrad to achieve better results. Adam is one of the few alternatives to the SGD so it will be one of the optimizers used on testing the models.

3. DATA PREPARATION

3.1 DATA COLLECTION

Since this study is a part of an ongoing project there was already a dataset and it consists of 200 class Turkish food. Also, more data is collected constantly from various sources during this study to create 31 class ingredient dataset. In addition to this, a web application "Deep Predictor" was created for collecting more data but due to time limitations, it was never used publicly.

3.1.1 Web Application "Deep Predictor"

The Deep Predictor is an open-sourced web application written in pure python that supports multiple deep learning libraries, multiple models running at the same time, and can run multiple predictions simultaneously. It is an easy to configure and easy to run application that is designed for collecting more images.

First, it was only planned to build a simple application for a single model for predicting images. But It is designed flexibly for most image predicting tasks that use deep learning. Because of that during the process of building the application, it is decided to make the application an open-sourced project for everyone to use as a backend for deep learning image prediction.

3.1.1.1 Example scenario

In an example scenario, if a person wants to use the application for predicting an image, he/she chooses an image from existing images from the application interface or if the client is on mobile a device, he can also take a new image for predicting. After choosing the image, he then can choose the model that he wants to predict with. (front-end model names can be modified for more user-friendly names while keeping the original model names at the back-end). After that, by simply pressing the predict button one can get the

prediction results as a good-looking bar chart. During this process, the Deep Predictor saves the image that has been sent by the user for later training.

3.1.1.2 Usage of the "Deep Predictor"

3.1.1.2.1 Configuring the "Deep Predictor"

Configurations are the biggest part that makes Deep Predictor flexible and allows us to run all of our DNNs without too much effort. Configuration of the Deep Predictor is done by configuration files. Deep Predictor has a main configuration file that has to be prepared before running it for production. The details about this file can be found on the GitHub repository of the project. After the main configuration file is prepared individual configuration files for each model that wanted to be deployed have to be created.

3.1.1.2.1.1 Predictor configuration files

A configuration file template is provided on the Deep Predictor GitHub repository for preparing individual configuration files. These files hold all the information that is required for Deep Predictor to run the model properly. To give an example, that information can be the location of the model's weight file, image size for Deep Predictor to resize the images to fit the model properly on prediction time, the name of the model, the back-end library that used for training the model and other critical information like these.

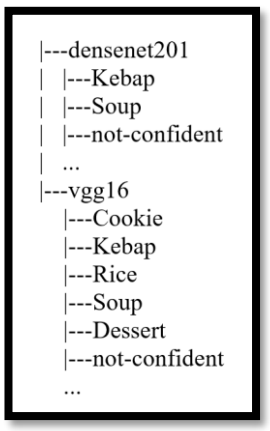
3.1.1.2.2 Running the "Deep Predictor"

The Deep Predictor is designed to be run easily on every machine that has the proper libraries installed, it is written on pure python so it can run almost all devices that can run python. In addition to that multiple versions of supported deep learning libraries are tested and documented for ones who want to run the Deep Predictor. After configuration steps are completed Deep Predictor can run both on development and production easily with a single command.

3.1.1.3 Saved images

Deep Predictor folders images by the name of the predicted class that is provided during the configuration. This way is implemented to decrease the work after collecting images since all the images are in their proper places for later training tasks. Also, since Deep Predictor uses different configuration files for each model that is running, it is possible to separate the saving locations of all the models. This is also useful to see which models are doing fine and which are doing bad, it is possible to understand the performance of the models simply by looking at the prediction folders for individual models for misclassifications. In addition to that, a threshold can be configured from each model's configuration file, with this threshold Deep Predictor can save the images that have lower accuracy than the threshold to the "not-confident" folder. This is also added to decrease the work after collecting the images, since there will be useless images sent by users it is easier for us to clean useless images if a big portion of them already filtered.

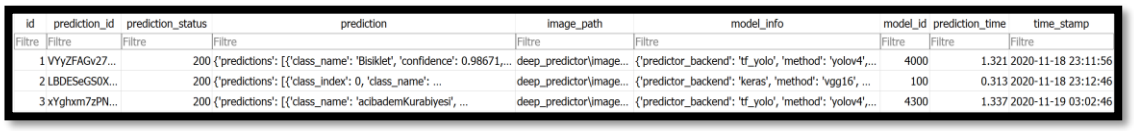
Figure 3.1: Example save structure



3.1.1.4 Database

The database of the Deep Predictor contains many useful information about the predictions. Such as which model with which configuration is predicted the image, the time stamp of the prediction, total prediction time, the full result of the prediction. These can be used for analyzing the model performances.

Figure 3.2: Database preview



id	prediction_id	prediction_status	prediction	image_path	model_info	model_id	prediction_time	time_stamp
Filtre	Filtre	Filtre	Filtre	Filtre	Filtre	Filtre	Filtre	Filtre
1	VYyZFAGv27...	200	{'predictions': [{'class_name': 'Bisiklet', 'confidence': 0.98671,...	deep_predictor\image...	{'predictor_backend': 'tf_yolo', 'method': 'yolov4',...	4000	1.321	2020-11-18 23:11:56
2	LBDESeGS0X...	200	{'predictions': [{'class_index': 0, 'class_name': ...	deep_predictor\image...	{'predictor_backend': 'keras', 'method': 'vgg16', ...	100	0.313	2020-11-18 23:12:46
3	xYghxm7zPN...	200	{'predictions': [{'class_name': 'acibademKurabiyesi', ...	deep_predictor\image...	{'predictor_backend': 'tf_yolo', 'method': 'yolov4',...	4300	1.337	2020-11-19 03:02:46

3.1.1.5 Logs

A robust logging system implemented in the Deep Predictor, it is possible to detect any warnings or errors with time stamps and error's stack trace. All the logs are located under the logs folder for convenience.

Figure 3.3: Log example 1

```
[flask][INFO] 2020-12-30 07:34:20 creating flask app
[flask][WARNING] 2020-12-30 07:44:11 prediction_id is not provided
[flask][INFO] 2020-12-30 08:00:13 flask_app using cfg file on path:deep_predictor/cfg/deep_predictor_dummy.cfg
```

Figure 3.4: Log example 2

```
[prediction_thread][INFO] 2020-10-20 23:15:03 prediction started on thread
[prediction_thread][WARNING] 2020-10-20 23:15:03 prediction failed with status 500
```

3.1.1.6 API

The Deep Predictor has a rest-style API for prediction results. Even though it has a front-end Deep Predictor's primary focus was always the back-end, so to support multiple front-ends that could be designed, later on, a familiar API standard is used. With this convenient API method, anyone who uses Deep Predictor can create a native application for mobile devices or another front-end web interface.

Figure 3.5: Successful API response example


```
{
  "model_info": {
    "method": "vgg16",
    "model_id": 100,
    "predictor_backend": "keras"
  },
  "prediction_id": "XC9ZNaDT2nfFaORUIeXSBims3WcdLJRS",
  "prediction_status": 200,
  "prediction_time": 1.32,
  "predictions": [
    {
      "class_index": 6,
      "class_name": "soup",
      "confidence": 0.97628
    },
    {
      "class_index": 4,
      "class_name": "apple",
      "confidence": 0.02319
    }
  ]
}
```


3.1.1.6.1 API usage

The API has multiple endpoints for making front-end development easier. Also, it is possible to change the names of the endpoints from the configuration file. With the "predictors" endpoint developer can get the required model information for displaying model names to users and with the "prediction_id" endpoint when the needed prediction id is provided developer can get a very detailed prediction result for the requested prediction. The returned prediction result has multiple useful information about the prediction for developers. Such as prediction status, the most confident predictions, confidence levels of those predictions, class name of predicted objects, prediction method, total prediction time, and more also if this is a detection task bounding boxes of the found images are also provided from this endpoint. All of the usage details of the API are documented with examples on the Deep Predictor's GitHub page.

3.1.1.7 Application tests

3.1.1.7.1 Unit tests

Multiple unit tests implemented for Deep Predictor. All the critical parts have individual unit tests. Such as, the "predictors", prediction thread, and a test for database functions

that have their own database. Other than that, more complex parts have also tests that use multiple unit tests to achieve the desired test.

3.1.1.7.2 Stress test

A prediction generator class is implemented to be used in stress tests. The prediction generator class can get test images from different directories, it also can check the test API to see which “predictors” are running. With using this class, a stress test is implemented it chooses multiple images randomly and sends those images to random “predictors” using multithreading to achieve fake stress on the system. This test can be used to test the system's behavior under pressure.

3.1.1.7.3 Dummy predictors

The purpose of “dummy predictors” is to try the system without installing any deep learning libraries. Since those libraries have a lot of dependencies it could take time to install them, not only that but also starting the system with real models depending on the number of models that wanted to be deployed can take more than 5 minutes. It is not common to encounter simple configuration errors during first tests so it is more convenient to set up everything with “dummy predictors” and run the real ones later.

3.1.1.7.4 Test API

Test API has separate endpoints from the regular API. The usage of the test API is testing the health of the system during production. Since all the images sent by users are saved by the Deep Predictor, testing the system via sending images would contaminate the collected image folders to prevent this test API is implemented. Test API has its own “predictors” for testing and those models can save different directories or can be modified to not save all. Those “test predictors” are not visible by the front-end and only usable by the developer-defined API key. By using the test API an example periodic test is created, this test tries to make a prediction every 1 hour to ensure everything is running properly else it e-mails to desired e-mail about the malfunction.

3.1.1.7.5 Front end

A simple front-end is implemented, this front-end has automated rescaling via CSS so it is also usable on most mobile devices. In the Image upload page, a google captcha is added to the form to prevent possible abuse or overload to the server. On the results page, animated bar chars were implemented with vanilla JavaScript to make results more user-friendly. Finally, as mention in the API section, Deep Predictor's API is designed flexible enough for using different front-ends without touching the backend code.

3.2 DATA ORIENTATION

Dataset orientation mostly done with self-made tools, to work fast with different environments most used functions are published as a python library.

3.2.1 Imagepreprocessing

During this study and the project that this study is a part of, multiple DNN architectures and multiple deep learning libraries were used. "Imagepreprocessing" is an open-sourced small library designed for speeding up the dataset preparation and model testing steps for deep learning on various libraries, it is a toolbox for this project. Most of the dataset orientation operations applied with this library. The library is published on the python package index "PyPI", and got some attention during the writing process of this project the latest version is 1.7.2, and it is downloaded over 25000 times according to "Pepy" website.

3.2.1.1 Functions of "Imagepreprocessing"

3.2.1.1.1 Darknet functions

Darknet is the framework that the original YOLO algorithm's authors' wrote Redmon (2017), but today a fork of this framework is continued with the community effort.

Darknet functions is a sub-function category of the Imageproprocessing that is created to be used with the darknet framework, especially the YOLO algorithm.

3.2.1.1.1.1 Create training data yolo

This function uses a directory of images as a parameter to create all the required files to train the YOLO algorithm. Those files can be time-consuming to create by hand every time.

3.2.1.1.1.2 Yolo annotation tool

This function is a tool for making the annotation files that the YOLO algorithm needs to run the training. Annotations are the bounding boxes of the objects that wanted to be detected by the YOLO algorithm. Those boxes have to be created by hand for each image, and YOLO uses a special format for the bounding box coordinates. This function uses OpenCV to show the bounding boxes that are drawn by the user and automatically converts those bounding boxes' coordinates to YOLO's bounding box format and saves them to a file to be used by the YOLO algorithm while training. This function is also added to GitHub as a separate project for people who want to use the annotation tool separate from the Imagepreprocessing library.

3.2.1.1.1.3 Auto annotation by random points

Our dataset mostly consists of single class food images, those images are cropped to be on the middle of the image and they are not required to be labeled by hand. But YOLO algorithm was designed to work with bounding boxes, this function was created to overcome this problem during the tests. This function automatically creates annotation files for images, using a selected set of coordinates.

3.2.1.1.2 Keras functions

Keras is a deep learning framework written in python by François Chollet. Later it is added into the TensorFlow project that is developed by Google. "Keras functions" is a sub-function category of the Imageproprocessing that is created to be used with the Keras framework.

3.2.1.1.2.1 Create training data keras

This function uses a directory of images to create a train-ready image collection object to be used by any CNN model that is created using the Keras framework. To be used in the training images has to go through some operations some of those are; reading images from the directory, shuffling, splitting as train-test, resizing, labeling, one-hot encoding, and converting to an array. This function does these operations in one line, it also can save the output as a single file for use on multiple tests later on.

3.2.1.1.2.2 Make prediction keras

This function is used to test multiple images directly from a directory. Some operations are required for testing as training, same as “create_training_data_keras” this function prepares multiple images in a single line.

3.2.1.1.3 Utilities

Utilities are other useful functions on the Imagepreprocessing library that can be used during training or testing models. Some examples for those functions are; “create_confusion_matrix” for creating confusion matrixes, “create_history_graph” for creating a graph for training history of a model, “remove_class_from_annotation_files” for removing a specific class from all YOLO annotation files in a directory, “count_classes_from_annotation_files” for counting object occurrences on YOLO annotation files in a directory.

3.2.2 Image Cropping Tool

Due to existing tools not provide the required capabilities, the image cropping tool is implemented. Unlike existing cropping tools this one is designed to be fast instead of unnecessarily precise for our study, fast image cropping is the key. The image cropping tool allows us to walk inside a directory that is full of images, and crop required parts of all the images without any extra save or image path selection. This increases the cropping speed and decreases downtime between each cropping task. Since this tool is only designed to be used by the developers, instead of buttons keyboard shortcuts are used for activating operations. The tool is implemented in python and OpenCV library used for image handling.

4. EXPERIMENTAL STUDIES AND RESULTS

4.1 MODEL TESTS WITHOUT CONTEXT INFORMATION

Various models are tested to decide the best models to be used with the improvement methods.

4.1.1 Dataset

Using our food dataset, 50 class and 100 class datasets were created for these tests. On all the tests the datasets are divided as 80 percent train and 20 percent test. Datasets shuffled before every test.

4.1.2 Preprocessing

Common data augmentation techniques were used during the tests. Those augmentations are; rotation, width and height shift, horizontal flip, zoom, shear, brightness chance, channel shift. Vertical flip is not used since an upside-down food plate is not expected to be detected. The rotation range was kept between 30 to 45 degrees. Brightness change set to 0.1 to 0.5 of the images.

Table 4.1: Data augmentation parameters

Rotation range	45
Width shift range	0.2
Height shift range	0.2
Horizontal flip	True
Zoom range	0.3
Shear range	0.2
Brightness range	0.1, 0.5
Channel shift range	0.5

4.1.3 Preliminary Test

A preliminary test is conducted with various models before more in-depth tests of the models.

4.1.3.1 Model

Models used in these tests are ResNetV2, InceptionV3, Xception, VGG16, DenseNet201 and DenseNet121. All of the models were tested on the 50-class dataset with SGD and 0.001 learning rate. Before testing, the last layers of the models changed with 50 to match the dataset and the SoftMax function was added to the last layer to get a classification result.

4.1.3.2 Training environment

Most tests run on the Google Colab platform, Colab provides free GPU for limited time usage and gives random GPUs on every login. Google Colab offers 3 different Nvidia GPU models those are; Tesla K80, Tesla T4, and Tesla p100. Some of the models are incapable of running on GPUs that have lover VRAM. To get the GPUs that have higher VRAM to run some of the models Colab is restarted until the desired GPU is given.

4.1.3.3 Result

After the tests completed DenseNet201 was the most successful model with 78 percent accuracy, it is followed by the other DenseNet model that is DenseNet121 with 76 percent accuracy. VGG16 stuck with 1 percent for a long time, it started the learn later on but it took a lot longer for VGG to train.

Figure 4.1: VGG16 training history

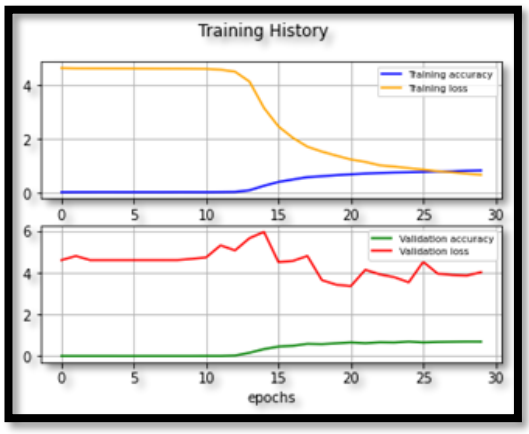


Figure 4.2: DenseNet201 training history

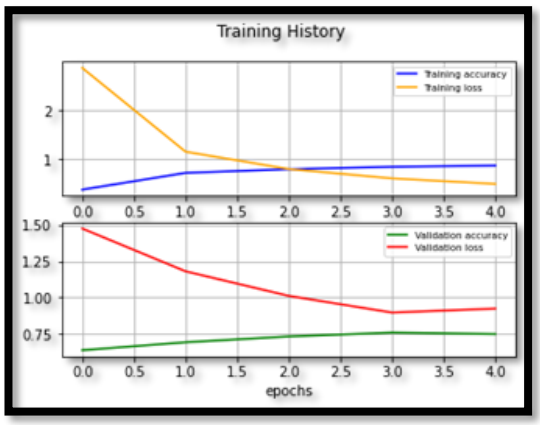


Table 4.2: Preliminary test results

DenseNet201	78%
DenseNet121	76%
ResNetV2	70%
InceptionV3	70%
VGG16	69%
Xception	65%

4.1.4 DenseNets

Based on the previous successes on the preliminary tests the DenseNets tested more in-depth with these tests.

4.1.4.1 Model

DenseNet201, DenseNet169, DenseNet121 are the versions of DenseNets that are used in this test. To use the DenseNets with these datasets, unit numbers of the last layers of the networks are changed to match with the datasets' class count. The SoftMax activation function is added to the last layers of the networks.

Other than SGD Adam, rmsprop, and adagrad optimizers are tested with DenseNets too. To make a better observation all test results calculated as top1, top3, top5, top10 accuracies, and also losses are saved.

4.1.4.2 Training environment

Google Colab is tested first but because of the VRAM problems and constant crushes of the Colab another platform searched for these tests. Google Cloud Platform (GCP) is a paid cloud service, it has a category called "compute engine" these engines can be modified with desired hardware. The first year Google gives 300 dollars of usage for free to the existing Google account owners. To use GPU on the GCP it should be requested first, Google mostly replies to the requests in 2 days, GPU request has to be done for each region separately. GCPs free trial is used to train these models. A powerful machine is created to speed up the training process. 2 Nvidia Tesla V100s are used on the machine, with multiple models training at the same time, training of each model took between 1 and 2 hours.

4.1.4.3 Result

Table 4.3: 50 Class DenseNet test results

50 class	DenseNet201	DenseNet169	DenseNet121
SGD lr=0.001, mmntm=0.9	top1: 0.8120 - top3: 0.9333- top5: 0.9600 - top10: 0.9867 - loss: 0.6781	top1: 0.8387 - top3: 0.9360- top5: 0.9653 - top10: 0.9827 - loss: 0.6612	top1: 0.8147 - top3: 0.9307- top5: 0.9573 - top10: 0.9827 - loss: 0.6983
SGD lr=0.001, nesterov	top1: 0.8307 - top3: 0.9480- top5: 0.9747 - top10: 0.9840 - loss: 0.5746	top1: 0.7920 - top3: 0.9027 - top5: 0.9467 - top10: 0.9747 - loss: 0.7800	top1: 0.8200 - top3: 0.9347 - top5: 0.9560 - top10: 0.9800 - loss: 0.6878
Adam	top1: 0.5933 - top3: 0.7760- top5: 0.8547 - top10: 0.9307 - loss: 1.7694	-	-
RMSprop	top1: 0.5013 - top3: 0.7307- top5: 0.8053 - top10: 0.8947 - loss: 2.1987	-	-
Adagrad	top1: 0.8400 - top3: 0.9413- top5: 0.9747 - top10: 0.9853 - loss: 0.6177	top1: 0.8253 - top3: 0.9253 - top5: 0.9560 - top10: 0.9827 - loss: 0.6596	top1: 0.8160 - top3: 0.9360 - top5: 0.9560 - top10: 0.9787 - loss: 0.6792

Table 4.4: 100 Class DenseNet test results

100 class	DenseNet201	DenseNet169	DenseNet121
SGD lr=0.001, mmntm=0.9	top1: 0.8487 - top3: 0.9500 - top5: 0.9729 - top10: 0.9900 - loss: 0.5041	top1: 0.8047 - top3: 0.9320 - top5: 0.9567 - top10: 0.9827 - loss: 0.8041	top1: 0.8047 - top3: 0.9267 - top5: 0.9587 - top10: 0.9800 - loss: 0.8000

All DenseNets reached top1 accuracy of 80 percent or better. DenseNet201 is the most successful network, it reached 85 percent top1 accuracy on 100 class tests and almost 100 percent top10 accuracy. DenseNets also reach their potential quickly, they tend to reach 60 percent top1 accuracy on the second or the third epoch.

After these tests, transfer-learning and fine-tuning were tested with ImageNet weights. Multiple tests were conducted and during these tests, a different number of layers were made untrainable to fine-tune the model, but no improvements were achieved. ImageNet weights appear to be not working well with our dataset.

Table 4.5: 50 Class DenseNet fine tuning and transfer learning results

50 class	**DenseNet201 fine tuning-last 18 layers trainable**	**DenseNet201 transfer learning + Dense(1000) + Dropout(.3) + Dense(1000) + Dropout(.3) + Dense(100)**	**DenseNet201 transfer learning + Dense(100)**
SGD lr=0.001, mmntm=0.9	top1: 0.8040 - top3: 0.9267- top5: 0.9520 - top10: 0.9813 - loss: 0.8472	top1: 0.7973 - top3: 0.9147- top5: 0.9453 -top10:0.9787- loss: 0.8039	top1: 0.7827 - top3: 0.9107- top5: 0.9480 -top10:0.9787- loss: 0.9337

Table 4.6: 100 Class DenseNet fine tuning and transfer learning results

100 class	**DenseNet201 fine tuning-last 18 layers trainable**	**DenseNet201 transfer learning + Dense(1000) + Dropout(.3) + Dense(1000) + Dropout(.3) + Dense(100)**	**DenseNet201 transfer learning + Dense(100)**
SGD lr=0.001, mmntm=0.9	top1: 0.7773 - top3: 0.9227 - top5: 0.9520 - top10: 0.9753 - loss: 0.8924	-	top1: 0.7433 - top3: 0.8947 - top5: 0.9273 -top10:0.9633 - loss: 1.2241

4.1.5 EfficientNet

It is shown that EfficeientNets can achieve the best ImageNet top-1 accuracy. Because of its successes on the ImageNet competition, EfficientNets tested with our food dataset.

4.1.5.1 Model

Since EfficientNet is a relatively new architecture there was no implementation of it on the existing libraries. At the time it only exists on the tf-nightly library. Tf-nightly is the daily updated version of Google's TensorFlow library. It is a little unstable but according to Google, it works most of the time. On these tests, EfficientNet models have been used from the tf-nightly library.

EfficientNet family has 8 different models with each is better than the other. Those models are named simply by adding "B" and the number of the model at the end of the

name. Full list of existing models is; EfficientNetB0, EfficientNetB1, EfficientNetB2, EfficientNetB3, EfficientNetB4, EfficientNetB5, EfficientNetB6, EfficientNetB7.

Like all other tests, the last layers of all models are modified for the task. Same optimizers were used as the DenseNet tests (SGD, Adam, rmsprop, and adagrad). Again top1, top3, top5, top10 accuracies with loss values are calculated on all the tests for better evaluation.

4.1.5.2 Training environment

The same machine on the Google Cloud Platform (GCP) used to train these models as the DesnseNet test. 2 Nvidia Tesla V100s managed to train the models in 1 to 2 hours each with multiple models running at the same time.

4.1.5.3 Result

EfficientNetB0, EfficientNetB1, EfficientNetB2, and EfficientNetB3 were not trained properly with any of the optimizers, they also tested with different learning rates but none of them managed to pass the 10 percent so they are not used on the remaining tests.

Table 4.7: 50 Class EfficientNet test results

50 class food	EfficientNetB7	EfficientNetB6	EfficientNetB5	EfficientNetB4
SGD lr=0.001, mmntm=0.9	top1: 0.5147 - top3: 0.6947 - top5: 0.7853 - top10: 0.8893 - loss: 1.9115			
SGD lr=0.01, nesterov	top1: 0.5600 - top3: 0.7387 - top5: 0.8053 - top10: 0.8947 - loss: 1.7674			
Adam lr=0.01	top1: 0.5947 - top3:0.7853 - top5:0.8427 - top10: 0.9253 - loss: 2.4126	top1: 0.0427 - top3: 0.0987 - top5: 0.1507 - top10: 0.2813 - loss: 4.6463	top1: 0.0640 - top3: 0.1320 - top5: 0.1947 - top10: 0.3040 - loss: 4.9932	
Adam lr=0.0001	top1: 0.7480 - top3: 0.8787 - top5: 0.9133 - top10: 0.9387 - loss: 1.8931	top1: 0.6613 - top3: 0.8280 - top5: 0.8947 - top10: 0.9467 - loss: 1.2866	top1: 0.7227 - top3: 0.8573 - top5: 0.9000 - top10: 0.9493 - loss: 1.2044	top1: 0.2013 - top3: 0.4080 - top5: 0.5293 - top10: 0.7093 - loss: 3.8200

RMSprop	top1: 0.5173 - top3: 0.6960 - top5: 0.7587 - top10: 0.8507 - loss: 1.9645	top1: 0.5387 - top3: 0.7320- top5: 0.7973 - top10: 0.9000 - loss: 2.0277	top1: 0.6787 - top3: 0.8373 - top5: 0.8787 - top10: 0.9373 - loss: 1.6196	top1: 0.1760 - top3: 0.3560 - top5: 0.4293 - top10: 0.5907 - loss: 5.4181
RMSprop lr=0.0001, mmntm=0.9	top1: 0.5173 - top3: 0.7080 - top5: 0.7853 - top10: 0.8747 - loss: 2.3280			
Adagrad	top1:0.4213 - top3: 0.6480 - top5: 0.7360 - top10: 0.8467 - loss: 2.1620			

Other than EfficientNetB4 the rest of the EfficientNet models resulted in mostly mode than 60 percent on their best results. One exception was the EfficientNetB7 model with 74 percent top1 accuracy on the 50 class tests but when the test is repeated it is observed that the results were inconsistent with all EfficientNets. On the repeated tests the most successful models were EfficientNetB5, EfficientNetB6, EfficientNetB7 with results between 55 to 70 percent top1 accuracies.

On the tests, Adam optimizer produced better results with EfficientNet than the SGD, because of that mostly Adam has used the rest of the tests.
On the 100 class tests, the best-resulted models used and those models resulted in between 60 to 65 percent top1 accuracy.

Results with EfficientNets were lower than expected especially when their ImageNet successes were considered. Because of this more tests with EfficientNet is conducted.

On all of the new tests EfficientNetB7 is used, transfer-learning and fine-tuning with ImageNet weights are tested. On these tests different number of layers made untrainable and the rest of the network trained to get better accuracies. Also adding more layers at the end is tested on some of the tests. But none of the tests resulted in better accuracy than training the model from scratch.

Table 4.8: 50 Class EfficientNet fine tuning and transfer learning results

50 class food	**EfficientNet B7 + Dense(100)**	**EfficientNet B7 + Dense(1000) + Dense(1000)**	**EfficientNet B7 + Dense(120) + Dense(120)**	**EfficientNet B7 + Dense(4000) + Dense(1000)**	**EfficientNet B7 + Dense(1000) + Dropout(.3) + Dense(100)**	**EfficientNet B7 transfer learning only new layers trainable + Dense(1000) + Dropout(.3) + Dense(1000) + Dropout(.3) + Dense(100)**	**EfficientNet B7 fine tuning- last 20 layers trainable**
Adam lr=0.0001	top1: 0.6547 - top3: 0.8133 - top5:0.8627 - top10: 0.9307 - loss: 0.9996	top1: 0.3573 - top3: 0.5253 - top5: 0.6227 - top10: 0.7587 - loss: 3.4924	top1: 0.4160 - top3: 0.6133 - top5: 0.6880 - top10: 0.7920 - loss: 3.0776	top1: 0.1253 - top3: 0.1960 - top5: 0.2693 - top10: 0.3960 - loss: 8.5259	top1: 0.5080 - top3: 0.6747 - top5: 0.7493 - top10: 0.8347 - loss: 2.5654	top1: 0.5507 - top3: 0.7387 - top5: 0.8067 - top10: 0.8760 - loss: 1.7320	top1: 0.3667 - top3: 0.5227 - top5: 0.6040 - top10: 0.7333 - loss: 2.9719

This and the DenseNets failed transfer-learning attempts with the ImageNet weights can be explained by the difference between datasets. For transfer-learning to work, the previously used dataset for training the network has to be something similar to the dataset that will be used with the trained weights. Even though ImageNet is a great dataset for transfer-learning on many datasets, apparently our dataset differs a lot from it so that transfer-learning is not working as intended. In short, ImageNet is a huge dataset with a very high variety of objects, but our dataset is a small dataset with objects that are very similar to each other, this explains the failure of the transfer-learning.

4.1.6 YOLO

4.1.6.1 Dataset

The same dataset is used on YOLO tests with the previous tests, but YOLO requires different files such as configuration files and individual image paths that will be used on training. Also, it requires labels for all the images in the dataset as specially formatted for

YOLO. All those files and the labels for images were created with the Imagepreprocessing library that is explained in detail in the SECTION 3.2.1.

4.1.6.2 Model

YOLO is a very successful real-time detection algorithm it is available on the Darknet framework, today a fork of the original framework is continued with the community effort after the original authors discontinued the original version.

4.1.6.3 Training environment

Since Darknet is not a widely used framework it is harder to install. It is purely written in C and CUDA so it has to be compiled before using it.

Our school provided azure cloud computers for deep learning tests. One of those computers used on YOLO test. The computer has 2 Nvidia Tesla M60s. After installing the required CUDA drivers to the machine Darknet is compiled and prepared for usage on the tests.

Like mentioned before Darknet is not a very popular framework, because of that there were a lot of GPU-related problems during the test, but some workarounds were found for those issues, and tests are successfully conducted.

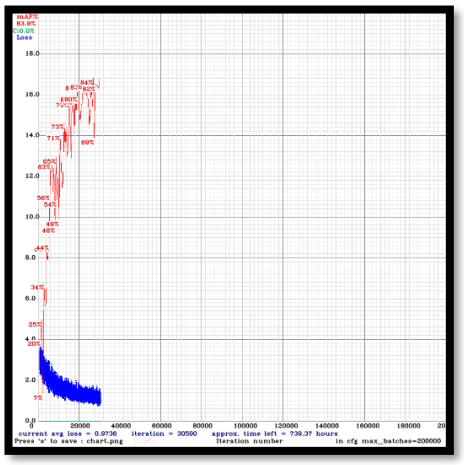
It took around 3 days to reach 30000 iterations when the YOLOs default train configurations are used. The suggested iteration count for 100 classes was 200000 but this number is calculated considering that is a detection task. When bounding boxes are set to the size of the image to use the algorithm for just classification, it is observed that training goes a lot faster. It is shown in the graph MAP value was not increasing for over 10000 iterations.

4.1.6.4 Result

Since YOLO is a detection algorithm, by default it calculates its accuracy with the MAP value. But the MAP value includes also bounding box accuracy and since all boxes are set to maximum size on the training set, results of the MAP value go higher.

When the accuracy for the YOLO calculated by hand using the test images it resulted in 63 percent for the 100 class tests.

Figure 4.3: YOLO Training Graph (MAP and Training loss)



4.2 RESOLUTİON TEST

This test is conducted to find the optimal image resolution to be used on the first fusion test.

For regular images, higher resolution always works better on deep learning, since more information is stored on higher resolution images. But ingredient images are different, they cropped from a dataset that already has small resolution images. That makes ingredient images extremely small and resizing those images to even 64x64 means that stretching the image. This means that increasing the resolution for the ingredient images just copies the information that the image already has and pasting it again to the image. Therefore, resizing to a higher resolution might not affect accuracy.

4.2.1 Dataset

A new dataset is required for this test. Our ingredient dataset has 31 classes of ingredients and they appear on 39 different main food classes. Those ingredients are "Ceviz", "Kasar", "Badem", "Yumurta", "Bezelye", "Findik", "Havuc", "Limon", "Dereotu", "Maydanoz", "Kabak", "Patlican", "Yogurt", "Domates", "Mantar", "Salatalik", "Turp", "Roka", "Sogan", "HindistanCevizi", "Kaymak", "SamFistigi", "FirindaPatates", "Kofte", "KusbasiEt", "PatatesKizartmasi", "KupSeker", "SoganPiyazi", "Biber", "BulgurPilavi", "PirincPilavi".

By using the cropping tool that is mentioned in SECTION 3.2.2 ingredient images are cropped. Those cropped images created the 31-class texture dataset to be used in this test. Our existing ingredient dataset has already small images, and when these images cropped to form the texture dataset image sizes decreased even more.

4.2.2 Model

Keras framework and TensorFlow back-end used to train the models on this test. Tested models are DenseNet201, DenseNet169, DenseNet121, VGG19, VGG16, ResNet152V2, ResNet101V2.

4.2.3 Training Environment

All models trained on Google Colab with different Nvidia GPUs.

4.2.4 Results

For all results top1, top3, top5, and top10 accuracies with the loss values for the best runs are calculated for better assessments of the models.

4.2.4.1 DenseNet

DenseNet models run with Nvidia Tesla p100 GPU on Google Colab. Each test took between 1 to 3 hours, depending on the load Colab can slow down CPU speed. DenseNet models tested different resolutions of the images and also with different optimizers to find the best accuracy.

Table 4.9: DenseNet 31 class ingredient texture test results

31 class ingredient texture	**DenseNet201**	**DenseNet169**	**DenseNet121**
SGD 32x32x3	top1: 0.4413 top3: 0.7207- top5: 0.8347 - top10: 0.8993 - loss: 2.4401	top1: 0.6347 top3: 0.8387- top5: 0.8933 - top10: 0.9200 - loss: 1.7842	top1: 0.6440 top3: 0.8393- top5: 0.8847 - top10: 0.9213 - loss: 1.8864
SGD 64x64x3	top1: 0.7440 - top3: 0.8767 - top5:0.9020 - top10: 0.9240 - loss: 1.5368	top1: 0.7500 - top3: 0.8827 - top5: 0.9093 - top10: 0.9260 - loss: 1.4839	top1: 0.7400 - top3: 0.8787 - top5: 0.9020 - top10: 0.9233 - loss: 1.5216
SGD 128x128x3	top1: 0.8033- top3: 0.8933 - top5: 0.9160 - top10: 0.9293 - loss: 1.1990	top1: 0.7740 - top3: 0.8873 - top5: 0.9180 - top10: 0.9287 - loss: 1.3918	top1: 0.7773 - top3: 0.8873 - top5: 0.9140 - top10: 0.9300 - loss: 1.4078
Adam 32x32x3	top1: 0.3807 top3: 0.6500 - top5: 0.7467 - top10: 0.8693 - loss: 2.9074	top1: 0.3420 top3: 0.6413 - top5: 0.7780 - top10: 0.8760 - loss: 3.3857	top1: 0.4280 - top3: 0.6633 - top5: 0.7600 - top10: 0.8633 - loss: 2.8816
Adam 64x64x3	top1: 0.6813 - top3: 0.8640 - top5: 0.9013 - top10: 0.9240 - loss: 2.0637	top1: 0.6460 - top3: 0.8613 - top5: 0.8967 - top10: 0.9240 - loss: 2.5629	top1: 0.6467 - top3: 0.8340 - top5: 0.8880 - top10: 0.9187 - loss: 2.1272

After the test, it is observed that the DenseNets gets higher results when the resolution is increased. Every image in the texture datasets has different resolutions but on average the

native resolution is closer to 32x32 but still, it gets the worst results. Also, the SGD optimizer with a 0.001 learning rate achieved better accuracy scores when it compared to the Adam optimizer with the same learning rate. The DenseNet201 model was the best scoring one out of all DenseNet models, it is also the overall best-resulting model on this test. The other two DenseNet models also got high results with SGD and higher resolution.

4.2.4.2 VGG

VGG models run with Nvidia Tesla T4 GPU on Google Colab. The test took roughly 1 hour each. VGG models are used to test the images with different resolutions and with different optimizers.

Table 4.10: VGG 31 class ingredient texture test results

31 class ingredient texture	**VGG19**	**VGG16**
SGD 32x32x3	top1: 0.6613 - top3: 0.8487 - top5: 0.8940 - top10: 0.9200 - loss: 2.8697	top1: 0.7093 - top3: 0.8700 - top5: 0.9047 - top10: 0.9227 - loss: 4.3052
SGD 64x64x3	top1: 0.7307 - top3: 0.8713 - top5: 0.9053 - top10: 0.9247 - loss: 4.5682	top1: 0.7553 - top3: 0.8753 - top5: 0.9113 - top10: 0.9267 - loss: 4.4225
SGD 128x128x3	top1: 0.7500 - top3: 0.8773 - top5: 0.9107 - top10: 0.9293 - loss: 5.0935	top1: 0.7547 - top3: 0.8800 - top5: 0.9107 - top10: 0.9280 - loss: 5.1126
Adam 32x32x3	top1: 0.3113 - top3: 0.5700 - top5: 0.6773 - top10: 0.8120 - loss: 3.7776	top1: 0.4193 - top3: 0.6793 - top5: 0.7740 - top10: 0.8753 - loss: 3.6429
Adam 64x64x3	top1: 0.3533 - top3: 0.5947 - top5: 0.7233 - top10: 0.8567 - loss: 3.8762	top1: 0.4027 - top3: 0.6827 - top5: 0.7873 - top10: 0.8820 - loss: 3.6488

Results show that VGG networks achieved closer accuracies with the DenseNet models. VGG models also produced better accuracies when higher resolution images are used. Another common result with DenseNets was SGD with a learning rate of 0.001 has better results with Adam with the same learning rate. VGG19 got slightly better results with higher resolution tests. VGG models got the best results after the DenseNets.

4.2.4.3 ResNet

ResNet models require more VRAM like DenseNets so they trained with Nvidia Tesla p100 GPU again on Google Colab. ResNets trained for around 1 to 3 hours. ResNet models are also tested with images that have different resolutions and with different optimizers.

Table 4.11: ResNet 31 class ingredient texture test results

31 class ingredient texture	**ResNet152V2**	**ResNet101V2**
SGD 32x32x3	top1: 0.3653 top3: 0.6167- top5: 0.7300 - top10: 0.8493 - loss: 2.3960	top1: 0.3827 top3: 0.6460 - top5: 0.7700 - top10: 0.8727 - loss: 2.3472
SGD 64x64x3	top1: 0.6107 - top3: 0.8147 - top5: 0.8660 - top10: 0.9073 - loss: 1.7215	top1: 0.5767 - top3: 0.7687 - top5: 0.8420 - top10: 0.9087 - loss: 2.0076
Adam 32x32x3	Nan	top1: 0.2747 - top3: 0.5267 - top5: 0.6713 - top10: 0.8293 - loss: 3.0403
Adam 64x64x3	top1: 0.5833 - top3: 0.8133 - top5: 0.8667 - top10: 0.9133 - loss: 2.2844	top1: 0.5513 - top3: 0.7780 - top5: 0.8580 - top10: 0.9173 - loss: 2.6339

After a low-resolution test with the ResNet model, it is observed that the accuracy was too distant from the DenseNet or VGG test. 128X128 resolution is not used to test ResNets since they are outclassed by the other networks on this dataset.

On all the model tests for the texture dataset, accuracy results were higher when a higher resolution image is used. Those results showed that just like regular-sized images, low-resolution images also produce better results with higher resolutions than their native resolutions. A similar and more detailed study is done in Kannojia (2018). They trained MNIST and CIFAR10 datasets with a custom model. Both datasets trained with different resolutions, those resolutions are 7x7, 14x14, 21x21, 28x28 for the MNIST and 8x8, 16x16, 24x24, 32x32 for the CIFAR10. In their study, it is also concluded that higher resolutions work better even if images are small.

Table 4.12: Result comparison with "Effects of Varying Resolution on Performance of CNN based Image Classification an Experimental Study"

Effects of Varying Resolution on Performance of CNN based Image Classification An Experimental Study (**Cifar 10**)		Our dataset (10 **class**)		Our dataset (**31 class**)	
Custom model	Accuracy (%)	DenseNet201	Accuracy (%)	DenseNet201	Accuracy (%)
16x16	42.33	16x16	-	16x16	-
32x32	87.52	32x32	73.17	32x32	44.13
64x64	-	64x64	75.05	64x64	74.40
128x128	-	128x128	77.78	128x128	80.33

DenseNet tests repeated with 10 class to compare with cifar 10 results, but since our dataset has not many samples per class, decreasing the class size also decreased the accuracy.

4.3 FUSION

4.3.1 Fusing Food Image with An Ingredient Image

There are different ways to use fusion, in this test fusion is done with 2 images. An ingredient image that is cropped from the main image is fused with the main image to achieve better accuracy.

4.3.1.1 Dataset

The texture dataset is used on these tests, ingredient images are cropped from the food dataset to form the texture dataset. The texture dataset is explained in more detail in the SECTION 4.2.1.

4.3.1.2 Model v1

The models that are tested during the previous test are planned to be tested in these tests. But since this fusion test is going to use 2 images 2 DNNs are required and known networks are too big to be used like this.

In order to test this fusion method, a smaller network is built from scratch. To make the network as simple as possible only 2 classes were used first. The new network is created based on the VGG networks, it has only 2 convolutional layers.

4.3.1.3 Result

First, the new model is tested without fusion, this test resulted in 80 percent accuracy. After that, 2 models were concatenated at the end to test the fusion, training was not properly worked on the first try. Batch size is modified to tweak the training process and using 8 as the batch size is worked. Also, steps are calculated per epoch for the new batch size for every image to be used on the training on each epoch. There were 120 images on the training set and batch size is set to 8 from this, optimal steps per epoch calculated as 15. When the fusion model is tested after training it resulted in between 91 to 93 percent accuracy. There was a slight improvement for a 2-class dataset.

4.3.1.4 Model v2

After the success of the 2 class tests, a 5-class test is also made. The model built for the 2-class test is tested first but it did not work properly. A new simple model built to be used on the 5-class test, this model has 4 convolutional layers.

4.3.1.5 Result

Before the fusion test again a regular test was conducted with the new model. The new model produced 63 percent accuracy. Results were not that high but since the goal of this

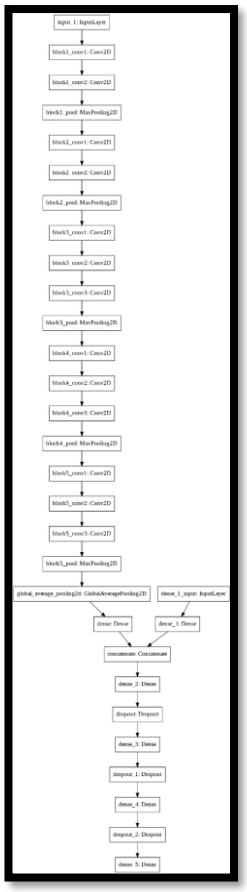
test is analyzing the effectiveness of the fusion high results were not necessary on this part.

The fusion network did not work right away again, batch size set to 10 this time for training and steps calculated accordingly. This time dataset had 300 images and with 10 images per batch steps calculated as 30 for every image to be used on each epoch. When the new model used with fusion on the 5-class dataset it resulted in 73 percent on its tests. There was again a small improvement for a 5-class dataset.

4.3.2 Fusing Food Image with Hand Crafted Ingredient Vector

In this test, fusion is used to fuse the food image data with handcrafted ingredient vector.

Figure 4.4: Fusion model with VGG



4.3.2.1 Dataset

The ingredient dataset is used for images, it has 39 classes. Those classes have 31 ingredients in them. Ingredient dataset explained more detailly in SECTION 4.2.1.

An ingredient vector structure is created for this test. For example, if there are 5 different ingredients in total, the vector for an image with no ingredients in it would look like this [0,0,0,0,0]. If 2 ingredients exist on the image the vector could look like this [0,1,0,1,0] depending on the ingredients.

Every food image is examined and the most common ingredients are saved for each food class. Then for every image, a corresponding ingredient vector is created and saved. To prevent wrong vector and image matches during the training image names on the dataset are examined and renamed if needed.

4.3.2.2 Model

The models that have been previously tested and achieved high scores are used in these tests. The most successful models were DenseNets, but VGG is also tested.

The CNN model and the ingredient vector concatenated at the end of the network, after the concatenation 3 fully connected layers added for the concatenated values to be merged properly.

4.3.2.3 Result

DenseNet201 is tested without fusion at first, it reached 82 to 83 percent accuracy on 10 epochs of training. When the fusion was applied with generated ingredient information DenseNet managed to get 87 percent accuracy with 25 epochs. DenseNet trained slower with fusion but achieved better accuracy.

Table 4.13: Fusion and no-fusion comparison for tested networks

39 class food	No fusion	Fusion
DenseNet201	top1: 0.8222 - top3: 0.9385 - top5: 0.9658 - top10: 0.9880 - loss: 0.3608	top1: 0.8774 - top3: 0.9887 - top5: 1.0000 - top10:1.0000 - loss: 0.2808
ResNet152V2	top1: 0.6855 - top3: 0.8462 - top5: 0.9162 - top10: 0.9487 - loss: 1.2044	top1: 0.7952 - top3: 0.8744 - top5: 0.9695 - top10: 1.0000 - loss: 0.9871
VGG16	top1: 0.7077 - top3: 0.8484 - top5: 0.9265 - top10: 0.9692 - loss: 1.1012	top1: 0.7949 - top3: 0.8662 - top5: 0.9744- top10: 1.0000 - loss: 0.9532

ResNet152V2 tested without fusion and the top result was 68 percent when trained for 16 epochs. When the fusion added result increased to 79 percent within 20 epochs. ResNet152V2 is also got better accuracies with fusion.

VGG16 is also tested without fusion and it produced 71 percent accuracy on 39 classes with 40 epochs. After fusion was added it reached 79 percent accuracy in only 25 epochs of training. VGGs accuracy is also increased with the fusion, and training time is drastically reduced.

Since ingredient vectors are created by hand, they were too “sterile”. This causes for 100 percent results on the top-5 and top-10 results. To solve this problem on the next test real ingredient values used form the study (Celik 2021).

Another study Khan (2019) Food items Detection and Recognition via multiple Deep Models is also showing that using fusion on food images improves the results significantly. In the study, they are using a combination of Food-11, Food-5k, Food-101, and UEC-Food datasets to create food - non-food set, and an 11-class food dataset. First, they are showing the results without fusion on the known state-of-the-art DNNs (VGG, AlexNet, GoogleNet, ResNet) and later they incorporate the fusion and compare the results. On the tests, it is shown that using fusion with food data improved their results.

4.3.3 Fusing Food Image with More Detailed Ingredient Vector

In this test, fusion is tested with the extracted ingredient information from (Celik 2021). In addition to ingredient vectors on the previous test, this test uses ingredients' relative positions to each other.

4.3.3.1 Dataset

Again, the ingredient dataset and the same ingredient vector structure used but, this time also ingredient occurrences used that is retrieved from the study (Celik 2021).

Since not all the food images contains the same amount of ingredients, food position vectors differ in size image to image. In order to compensate that empty parts padded with "0" zeros.
Ingredient relative positions uses ingredients' bounding boxes that are predicted by the YOLO algorithm, the information contains these columns "from_x, from_y, from_w, from_h, to_x, to_y, to_w, to_h"

4.3.3.2 Model

Same base models used with the previous test. The difference was on the fusion part, last test fused a food image and an ingredient vector but, on this test, there is also an ingredient position vector that has to be added to the fusion.

Ingredient position vector contains all the positions of the ingredients relative to each other and this makes every element of this vector is a 2d vector, since fusion has to be made with vectors that has the same dimensions position vector is flattened before fusing.

After the fusion 3 fully connected layers used at the end of the network to fuse the values properly.

4.3.3.3 Result

Since base networks tested without fusion on the previous test, those results used as a baseline also for this test.

Like all the other test DenseNet201 was the highest resulted one. Accuracy of the DenseNet201 was 82 percent before fusion. After the addition of the ingredient information DenseNet201's accuracy increased to 84 percent.

Table 4.14: New fusion and no-fusion comparison for tested networks

39 class food	No fusion	Fusion new
DenseNet201	top1: 0.8222 - top3: 0.9385 - top5: 0.9658 - top10: 0.9880 - loss: 0.3608	top1: 0.8402 - top3: 0.9452 - top5: 0.9867 - top10: 0.9987 - loss: 0.3120
ResNet152V2	top1: 0.6855 - top3: 0.8462 - top5: 0.9162 - top10: 0.9487 - loss: 1.2044	top1: 0.7426 - top3: 0.8744 - top5: 0.9485 - top10: 0.9738 - loss: 0.9982
VGG16	top1: 0.7077 - top3: 0.8484 - top5: 0.9265 - top10: 0.9692 - loss: 1.1012	top1: 0.7762 - top3: 0.8664 - top5: 0.9564- top10: 0.9720 - loss: 1.0532

ResNet152V2 is increased its results form 68 percent to 74 percent with fusion. VGG16 is also got an increase with context information and reached 76 percent accuracy.

The used ingredient vectors are closer to real life data on these tests, and due to the datasets size decrease, some of the results decreased when it compared to the second fusion test.

Since our ingredient dataset is unique, to make an accurate comparison with the literature, similar studies were searched. In the Rajayogi (2019) Indian Food Image Classification with Transfer Learning study, they tested their custom 20 class Indian food dataset with VGG, Inceptionv3, and ResNet. Their dataset is similar to ours. But we have 2 times more classes. They also incorporate transfer learning. In their study their best result with ResNet is 69 percent this result is close with our no fusion result. With fusion our model achieves better accuracies than that.

Table 4.15: Result comparison with "Indian Food Image Classification with transfer learning

Indian Food Image Classification with Transfer Learning		**Our dataset (39 class)**	
Method	**Accuracy (%)**	**Method**	**Accuracy (%)**
ResNet	**69.91**	**ResNet fusion**	**74.26**
Vgg16	**78.20**	**Vgg16 fusion**	**77.62**
Inception v3	**87.90**	**DenseNet fusion**	**84.02**

As shown in the results, our fusion results either better or close to the results of Rajayogi (2019), even though our dataset has twice as many classes.

5. LIMITATIONS AND FURTHER WORK

5.1 LIMITATIONS

On the first fusion test that is on the SECTION 4.3.1 custom models had to be built. This is because the previously tested well-known models become too large when two of them combined for the fusion to be tested on moderate hardware. Those custom models are not capable of classifying large number of classes. Improvements can be made for those models for testing the fusion's capability on larger number of classes.

Overall, more data for training is always awarded by DNNs, more food data would be resulted better training and test accuracy results. In addition, more ingredients can be added to the ingredient dataset. There are 31 ingredients on the dataset and those ingredients exists on 39 food classes. Since only visible ingredients can be used, number of ingredients are a little low. More ingredients can be extracted from existing food classes so that fusion can benefit more to the classification results.

5.2 FURTHER WORK

5.2.1 Priming and Pruning

Those are the proposed methods before starting this study. After research done for the study, it is concluded that fusion will be a more effective method for this project. It is because this project focuses more on adding external contextual information rather than using already learned information to improve. Because of this and also time limitations, the study mostly focused on fusion. In the Rosenfeld (2017) Priming Neural Networks article, it is shown that using pruning and priming increases the accuracy results of YOLOv2 by Redmon (2016) 1.4 percent for only pruning and 3.8 percent with both pruning and priming. But over the years DNN architectures got better and YOLOv3 Redmon (2018) exceeds those results without using pruning or priming. There was no

priming or pruning studies in recent years, and another study can be done for those methods.

5.2.2 Image Quality

In the SECTION 4.2 it is shown that using better quality images on the training improves the results. A separate study can be done for collecting better quality images for extracting better quality ingredients. Also, it is possible to collect ingredient images directly as high-quality images. But only collecting the ingredient for training purposes can cause worse training accuracies, since it is better for DNNs to train as close to real-life data as possible.

6. CONCLUSION

Preliminary tests with known CNN models showed that DenseNets works the best with the dataset. When more in-depth tests were done with DenseNets, they also showed the success of the DesneNet with the ingredient dataset.

EfficientNets were also tested, they did not result as high as DenseNets. They expected to be resulted high because of the ImageNet competition successes of them. After regular tests transfer learning is also tested with EfficientNets using ImageNet weights. Transfer learning also did not result well enough considering the ImageNet results. After these tests with EfficientNets, it is concluded that due to the dataset differences EfficientNets cannot produce good results with our dataset. That difference can be explained by this; ImageNet is a huge dataset with very different objects in it but our ser is the opposite of that, it is small compared to ImageNet and our food classes are looking like each other.

After regular model tests, a resolution test is done before testing the fusion. This test is done for finding the best resolution for the ingredient images that are cropped from our dataset. Test with different resolutions showed that using higher resolutions for training even if the new resolution is higher than the native resolution of the images results in higher accuracies.

Later, the first fusion test is conducted. This test is done by fusing an ingredient image that is extracted from a food image, and the food image itself during training. It was expected for DNN to learn ingredient occurrences on food images and got more accurate predictions. Since two DNNs are required for this test larger DNN modes caused problems with moderate hardware. Instead, a new model was created for this test, it is based on VGG but many layers were cut to fit two of them to the machine.

This model tested with and without fusion to examine the effects of the fusion on training results, and the fusion version resulted in higher accuracies than the non-fusion version of the model.

After that, another fusion test is done. In this test instead of expecting DNN to extract information from raw ingredient images, it is extracted by hand before training. This information is shaped as an ingredient vector. This vector is fused with a CNN that is given a food image the vector fused with the image contains the ingredient information in it. An advantage of this test to the first fusion test was being able to use the previously tested model, this is because the ingredient vector is a small text and does not require to be processed like an image. That makes the fusion less computationally expensive.

This test is also showed that using the ingredient information is increases the accuracy results of the DNNs. Since this fusion was less resource-heavy bigger CNN models were used on the tests, and they all improved by the fusion.

Finally, a third fusion test is done. In this test again an ingredient vector is used but this time instead of a hand-crafted vector, a vector that is extracted from the image with different contextual information methods is used, these methods are explained in more detail in the Celik (2021).

It is shown that using fusion to add contextual information to CNNs' increases the prediction accuracies of the models.

REFERENCES

Publications

Byerly, A., Kalganova, T., G., and Dear, I., 2020. A Branching and Merging Convolutional Network with Homogeneous Filter Capsules. *Researchgate*, [online] Available at: <https://www.researchgate.net/publication/338840076_A_Branching_and_Merging_Convolutional_Network_with_Homogeneous_Filter_Capsules> [Accessed 21 June 2021].

Cao, G., Xie, X., Yang, W., Liao, Q., Shi, G., and Wu, J., 2017. Feature-Fused SSD: Fast Detection for Small Objects. *Arxiv*, [online] Available at: <https://arxiv.org/abs/1709.05054> [Accessed 21 June 2021].

Deng, L., Chen, J., Ngo, C., Sun, Q., Zhang, Y. and Chua, T., 2021. Mixed Dish Recognition with Contextual Relation and Domain Alignment. *ieeexplore*, [online] Available at: <https://ieeexplore.ieee.org/abstract/document/9411739> [Accessed 21 June 2021].

Ge, W., Yang, S., and Yu, Y., 2018. Multi-Evidence Filtering and Fusion for Multi-Label Classification, Object Detection and Semantic Segmentation Based on Weakly Supervised Learning. *Arxiv*, [online] Available at: <https://arxiv.org/abs/1802.09129> [Accessed 21 June 2021].

Girshick, R., 2015. Fast R-CNN. *Arxiv,* [online] Available at: <https://arxiv.org/abs/1504.08083>. [Accessed 21 June 2021].

Han, F., Guerrero, R., and Pavlovic, V., 2020. CookGAN: Meal Image Synthesis from Ingredients. A*rxiv*, [online] Available at: <https://arxiv.org/abs/2002.11493> [Accessed 21 June 2021].

He, K., Zhang, X., Ren, S. and Sun, J., 2015. Deep Residual Learning for Image Recognition. *Arxiv*, [online] Available at: <https://arxiv.org/abs/1512.03385> [Accessed 21 June 2021].

Huang, G., Liu, Z., Van Der Maaten, L. and Weinberger, K., Q., 2016. Densely Connected Convolutional Network. *Arxiv*, [online] Available at: <https://arxiv.org/abs/1608.06993> [Accessed 21 June 2021].

Kannojia, S., P. and Jaiswal, G., 2018. Effects of Varying Resolution on Performance of CNN based Image Classification: An Experimental Study. *Researchgate*, [online] Available at: <https://www.researchgate.net/publication/328960034_Effects_of_Varying_Resolution_on_Performance_of_CNN_based_Image_Classification_An_Experimental_Study> [Accessed 21 June 2021].

Kawano, Y. and Yanai, K., 2013. Real-Time Mobile Food Recognition System. *Cv-foundation*, [online] Available at: <https://www.cv-oundation.org/openaccess/content_cvpr_workshops_2013/W03/html/Kawano_Real-Time_Mobile_Food_2013_CVPR_paper.html>. [Accessed 21 June 2021].

Khan, S., Ahmad, K., Ahmad, T. and Ahmad, N., 2019. Food items Detection and Recognition via multiple Deep Models. *Researchgate*, [online] Available at: <https://www.researchgate.net/publication/330312331_Food_items_Detection_and_Recognition_via_multiple_Deep_Models> [Accessed 21 June 2021].

Krizhevsky, A., Sutskever, I. and Hinton, G., E., 2012. ImageNet Classification with Deep Convolutional Neural Networks. *Researchgate*, [online] Available at: <https://www.researchgate.net/publication/267960550_ImageNet_Classification_with_Deep_Convolutional_Neural_Networks> [Accessed 21 June 2021].

Leon, B., 2012. Stochastic Gradient Descent Tricks. *Springer,* [online] Available at:<https://link.springer.com/chapter/10.1007/978-3-642-35289-8_25>. [Accessed 21 June 2021].

Lu, C., Krishna, R., Bernstein, M. and Fei-Fei, L., 2016. Visual Relationship Detection with Language Priors. *Arxiv,* [online] Available at: <https://arxiv.org/abs/1608.00187> [Accessed 13 February 2021].

Mao, J., Huang, J., Toshev, A., Camburu, O., Yuille, A. and Murphy, K., 2016. Generation and Comprehension of Unambiguous Object Descriptions. *Arxiv,* [online] Available at: <https://arxiv.org/abs/1511.02283>. [Accessed 21 June 2021].

Nagaraja, V., Morariu, V. and Davis, L., 2016. Modeling Context Between Objects for Referring Expression Understanding. *Arxiv,* [online] Available at: <https://arxiv.org/abs/1608.00525v1>. [Accessed 21 June 2021].

Öztürk Ergün, Ö. and Öztürk, B., 2018. An ontology based semantic representation for Turkish Cuisine. *Ieeexplore,* [online] Available at: <https://ieeexplore.ieee.org/document/8404617> [Accessed 21 June 2021].

Pato, L., V., Negrinho, R., and Aguiar, P., M., O., 2019. Seeing without Looking: Contextual Rescoring of Object Detections for AP Maximization. *Arxiv,* [online] Available at: <https://arxiv.org/abs/1912.12290> [Accessed 21 June 2021].

Rajayogi, J., R., Manjunath, G. and Shobha, G., 2019. Indian Food Image Classification with Transfer Learning. *Researchgate,* [online] Available at: <https://www.researchgate.net/publication/339904639_Indian_Food_Image_Classification_with_Transfer_Learning> [Accessed 21 June 2021].

Redmon, J. and Farhadi, A., 2016. YOLO9000: Better, Faster, Stronger. *ArXiv,* [online] Available at: <https://arxiv.org/abs/1612.08242> [Accessed 21 June 2021].

Redmon, J. and Farhadi, A., 2018. YOLOv3: An Incremental Improvement. *Arxiv*, [online] Available at: <https://arxiv.org/abs/1804.02767> [Accessed 21 June 2021].

Redmon, J. Divvala, S., Girshick, R., and Farhadi, A., 2015. You Only Look Once: Unified, Real-Time Object Detection. *Arxiv*, [online] Available at: <https://arxiv.org/abs/1506.02640> [Accessed 21 June 2021].

Ren, S., He, K., Girshick, R., and Sum, J., 2015. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *Arxiv*, [online] Available at: <https://arxiv.org/abs/1506.01497> [Accessed 21 June 2021].

Rosenfeld, A., Biparva, M. and Tsotsos, J., K., 2017. Priming Neural Networks. *Openaccess.thecvf*, [online] Available at: <https://openaccess.thecvf.com/content_cvpr_2018_workshops/w39/html/Rosenfeld_Priming_Neural_Networks_CVPR_2018_paper.html> [Accessed 21 June 2021].

Shrivastava, A., and Mulam, H., 2016. Contextual Priming and Feedback for Faster R-CNN. *Researchgate*, [online] Available at: <https://www.researchgate.net/publication/308277253_Contextual_Priming_and_Feedback_for_Faster_R-CNN> [Accessed 21 June 2021].

Shrivastava, A., Sukthankar, R., Malik, J., and Gupta, A., 2016. Beyond Skip Connections: Top-Down Modulation for Object Detection. A*rxiv,* [online] Available at: <https://arxiv.org/abs/1612.06851> [Accessed 21 June 2021].

Simonyan, K. and Zisserman, A., 2015. Very Deep Convolutional Networks for Large-ScaleImage Recognition. *Arxiv,* [online] Available at: <https://arxiv.org/pdf/1409.1556.pdf> [Accessed 21 June 2021].

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A., 2014. Going Deeper with Convolutions. *Arxiv*, [online] Available at: <https://arxiv.org/abs/1409.4842> [Accessed 21 June 2021].

Tan, M. and Le, Q., V., 2019. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. *Arxiv*, [online] Available at: <https://arxiv.org/abs/1905.11946v3> [Accessed 21 June 2021].

Tanno, R., Okamoto, K. and Yanai, K., 2016. DeepFoodCam | Proceedings of the 2nd International Workshop on Multimedia Assisted Dietary Management. *acm*, [online] Available at: <https://dl.acm.org/doi/abs/10.1145/2986035.2986044> [Accessed 21 June 2021].

Topouroglou, O., Chaita, T. and Kiourt, C., 2020. Deep Learning Platforms in Food Recognition | 24th Pan-Hellenic Conference on Informatics. *Acm*, [online] Available at: <https://dl.acm.org/doi/abs/10.1145/3437120.3437265> [Accessed 20 June 2021].

Touvron, H., Vedaldi, A., Douze, M., and Jégou, H.. 2019. Fixing the train-test resolution discrepancy. *Arxiv*, [online] Available at: <https://arxiv.org/abs/1906.06423> [Accessed 21 June 2021].

WU, W., 2009. Fast Food Recognition from Videos of Eating for Calorie Estimation - IEEE Conference Publication. *Ieeexplore*, [online] Available at: <https://ieeexplore.ieee.org/abstract/document/5202718>. [Accessed 21 June 2021].

Xie, Q., Luong, M., Hovy, E., Le, Q., V., 2019. Self-training with Noisy Student improves ImageNet classification. *Arxiv*, [online] Available at: <https://arxiv.org/abs/1911.04252> [Accessed 21 June 2021].

Yanai, K. and Kawano, Y., 2015. Food image recognition using deep convolutional network with pre-training and fine-tuning. *Ieeexplore*, [online] Available at:<https://ieeexplore.ieee.org/abstract/document/7169816> [Accessed 21 June 2021].

Yin, G., Sheng, L., Liu, B., Yu, N., Wang, X., and Shao, J., 2019. Context and Attribute Grounded Dense Captioning. *Arxiv*, [online] Available at: <https://arxiv.org/abs/1904.01410> [Accessed 21 June 2021].

Yu, L., Poirson, P., Ynag, S., Berg, A., C., and Berg, T., L., 2016. Modeling Context in Referring Expressions. *Arxiv*, [online] Available at: <https://arxiv.org/abs/1608.00272> [Accessed 21 June 2021].

Zhang, M., Tseng, C., and Kreiman, G., 2019. Putting visual object recognition in context. *Arxiv*, [online] Available at: <https://arxiv.org/abs/1911.07349> [Accessed 21 June 2021].

Zhang, V., Gong, L., Fan, L., Ren, P., Huang, Q., Bao, H., Xu, W., 2019. A Late Fusion CNN for Digital Matting. *Paperswithcode*, [online] Available at: <https://paperswithcode.com/paper/a-late-fusion-cnn-for-digital-matting> [Accessed 21 June 2021].

Zhang, Y., Jiang, H., Wu, B., and Fan, Y., 2019. Context-Aware Feature and Label Fusion for Facial Action Unit Intensity Estimation with Partially Labeled Data. *Researchgate*, [online] Available at: <https://www.researchgate.net/publication/339559240_Context-Aware_Feature_and_Label_Fusion_for_Facial_Action_Unit_Intensity_Estimation_With_Partially_Labeled_Data> [Accessed 21 June 2021].

Other Publications

Celik, M., E., (2021). Improving deep neural networks Extracting context knowledge from food Images. Thesis for the M.A. Degree İstanbul: Bahçeşehir University FBE.