Group Project - Cibus Enterprise

Convolutional Neural Networks ("CNNs")

Food Recognition Algorithm

Nova IMS Information Management School

Postgraduate Program in Enterprise Data Science & Analytics

Deep Learning Neural Networks 2022

4th February 2023

Ana Sofia Medeiros, Inês Pinto, Joana Fradinho and Manuel Santos

Abstract

The Cibus company will face a change in the technological landscape. Through the application of machine learning algorithms in the workplace, the owners' objective is to optimize the company's financial outcomes, the efficiency of the purchasing process and the customer experience. A process of image recognition will enable cost reduction and, as a result, maximize financial outcomes. The objective of this project is to accurately identify different types of food with the application of a Convolutional Neural Network model. Although the team considered several models in this analysis, the model that was able to obtain best results on the test dataset was the InceptionV3, a pretrained network, with an accuracy of 65,5%.

1 Introduction

Considering image recognition is one of the cutting-edge technologies that can be used to expedite the purchasing process, Cibus owners' objective is to use it to automatically identify the products chosen by the customer. After choosing the product, the consumer must confirm if the identified item is correct and manually submit the quantity of each item on an interactive monitor. In the end, the total is presented to the customer, as Cibus implemented a fixed price for unit. The purpose of this report is to present the deep learning algorithm chosen to apply in the creation of the image recognition engine.

Nowadays, Convolutional Neural Networks ("CNNs") are the state-of-the-art models for image classification since these models can learn and extract features from the input images, and then apply those patterns to classify them. However, some problems are usually associated

with this type of algorithms, such as overfitting. In the course of the work described in this report, some techniques were applied to avoid this common issue, namely early stop, that interrupts the training when a monitored metric has stopped improving and dropouts, which is a regularization technique, where some neurons are ignored, associated with a defined probability during the training process.

Moreover, transfer learning techniques also represent a turning point on the imagine recognition scope, as they allow to achieve better results as these models are already pre trained in large datasets. Therefore, the transfer learning techniques apply the knowledge from previous models to new ones, without starting the training process from zero, saving time and resources (Kandel, Castelli and Popovič 2020). Considering the abovementioned reasons, transfer learning models are widely used on the industry today.

In this work, the group followed a methodology that focuses on different stages, namely:

- **Dataset**: Includes the detail regarding the chosen dataset origin;
- **Data load and data preparation:** Describes the activities involved in the loading and preparation of the final dataset before the CNN algorithms were applied;
- **Model Development:** Describes the different models that were considered: baseline models designed from scratch and three pre-trained models, InceptionV3, Resnet50 and VGG16, as well as the tuning process for each case;
- **Evaluation:** Details how the model quality and effectiveness was evaluated;
- **Testing the model on unseen data:** Includes detail regarding the performed assessment of the model results on unseen data; and,
- **Conclusion:** Includes the final remarks regarding the algorithm and its application.

2 Dataset

The dataset used in the project was attained from the Kaggle repository. The Food 101 (Food 101 | Kaggle) dataset is divided in 101 categories, and each category is composed by 1.000 images of food. In total, the dataset is composed by 101.000 distinct food pictures.

3 Data load and preparation

The upload of the images was performed using a URL network downloader, in order to obtain the files directly from the URL and an archiving program to store the images in a single archive, that did the extract, the unzip and the print of the file names for all images automatically.

After importing the images that will be used in the project, it was needed to split the data between training (and validation) and test data, as required in deep learning models. This step was based on a function that creates the paths for the directory in which the training and test

data will be stored separately, allowing to differentiate the data types as well the food names. The training dataset contained 75.750 images belonging to 101 different classes, as presented in the image below.

Figure 1 – Visualization of a few elements of the training directory.

Both the training and the testing datasets presented the following distribution:

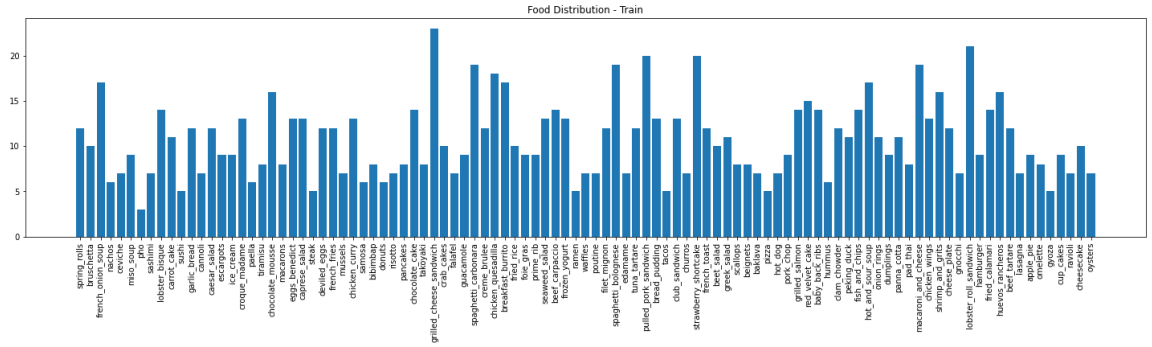

Figure 2 – Distribution of images per class on the training directory.

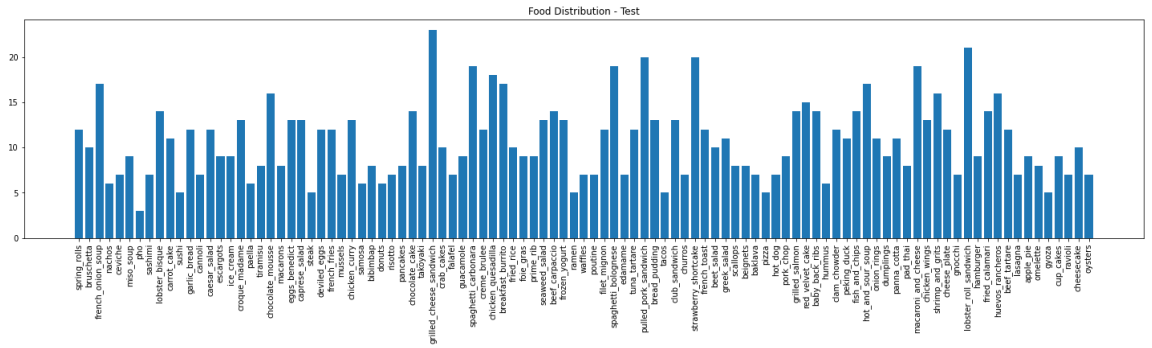

Figure 3 – Distribution of images per class on the testing directory.

In order to execute the data pre-processing step, the *ImageDataGenerator* was considered. The loading of the images was performed using one of the three predefined methods (i.e. *flow_from_directory*), that reads directly from a directory. With the beforementioned method,

the training and validation sets were created (with 15% of the data for validation) considering the training directory, as well as the test dataset from the test directory.

In the pre-processing step it was also applied the data augmentation step, which may be seen as the injection of previous knowledge about data patterns that are resistant to specific changes (Li, Kaan Tokgoz, Fukawa, and others 2021). This technique allows to increase the training dataset through the application of several transformations, with realistic variants, which prevents overfitting by enabling tolerance variations to the trained model.

The augmentation process, included on the *ImageDataGenerator* class, was performed by defining a set of possible transformations to these images, such as: specify the range in which to decrease/increase brightness, shift the width of the image 50% and random rotation by 90 degrees, to apply a 180-degree flip horizontally.

A few examples of output images from the augmentation process are presented in the next page:

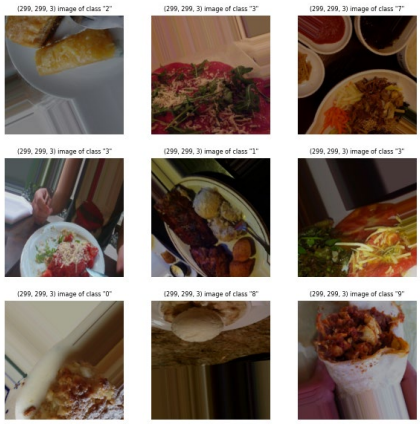

Figure 4 – Visualization of a few elements of the dataset after applying the data augmentation process.

It is also important to note that, for both training and validation subsets, the pictures size was resized to 224x224, to be aligned with some of the employed models used ahead (i.e. VGG16). Additionally, only the first 10 classes of the dataset (i.e. selected through alphabetic order) were considered, i.e. the 10 types of foods (i.e. apple pie, baby back ribs, baklava, beef carpaccio, beef tartare, beet salad, beignets, bibimbap, bread pudding and breakfast burrito). This cut was performed in order to optimize model performance.

Other important feature applied in the pre-processing stage was global normalization, where each pixel values were divided by 255. With this step, the computation becomes easier and faster.

4 Model Development and Tuning

Regarding the model development and tuning, the chosen approach was to start by defining one baseline CNN model and then create two additional CNN models with increasing complexity. Moreover, the group has also applied three distinct transfer learning techniques, i.e. VGG16, ResNet50 and Inception V3. Importantly, at this stage, all these models were applied to the train and validation datasets.

4.1. Optimizer and callbacks

All the tested approaches considered the use of an optimizer, which is a function that adapts the parameters of a neural network in order to reduce the value of the loss function, leading into a reduction of the overall loss and consequently improving the accuracy of the model. By that, there are different types of optimizer functions and each one should be tested according to the specific situation. There are two most common optimizers functions that perform well on this kind of situation: Stochastic Gradient Descent ("SGD") and the Adam (Adaptive Moment Estimation). While the first algorithm computes the gradient descent for the entire dataset and by that it is very computationally expensive, the second one performs a parameter update for each training data point. Regarding the Adam optimizer, this algorithm is the most used in the industry as it adapts its learning rates for each parameter and combines the advantages of SGDM in respect of the momentum and the scaling from RMSProp (Aremu, 2022). This way, the presented results consider the obtained results with Adams optimizer with a learning rate of 0.0001.

Additionally, to monitor and improve each model, the group considered three distinct keras callback techniques, as according to Keras documentation, callbacks are a "*set of functions to be applied at given stages of the training procedure*", and consequently allow to monitor and improve the model during training. Therefore, the callbacks contemplated in the project are:

- The *PlotLosses* technique, as it allows to visualize the updated graphs of loss and accuracy after each epoch. This is of relevance, as per research, loss and accuracy are two widely used performance metrics in Machine Learning models, as the loss value will imply how a model behaves after each training epoch and accuracy is used to measure the model's performance in an interpretable form;
- The Early stop technique was considered given it simplicity and popularity. This technique interrupts the training process as soon as the performance on the validation dataset decreases over a certain number of epochs (patience), allowing to reduce overfitting. The performance monitoring is done through the loss function. In this work we considered a patience of 5 epochs; and,

- The *ModelCheckpoint* technique was also implemented, allowing to save the model best and latest weights in a file after each epoch, in this case with a verbose equal to 1 (i.e., so it displays messages when the callback takes action).

4.2. Baseline models

The following describes the model architecture and obtained results for the three CNN models that were developed:

CNN Baseline Model 1:

- Conv2D layer: 64 filters of size (3,3) with input shape of (128,128,3)
- Conv2D layer: 64 filters of size (3,3)
- Max Pooling: MaxPool2D with (2,2) layers
- 1 Flatten Layer to transform the layers into 1 dimension
- 1 Dense Layer of 10 nodes with a softmax activation function

This model was compiled taking into consideration the categorical_crossentropy loss function with the Adam optimizer with a learning rate of 0.0001. Since the dataset is balanced, the metric used was the accuracy. This base model was performed for 15 epochs with an early stop of 5 epochs to avoid overfitting. After 15 epochs this model showed an accuracy of 41,5% in the training dataset and 41% in the validation.

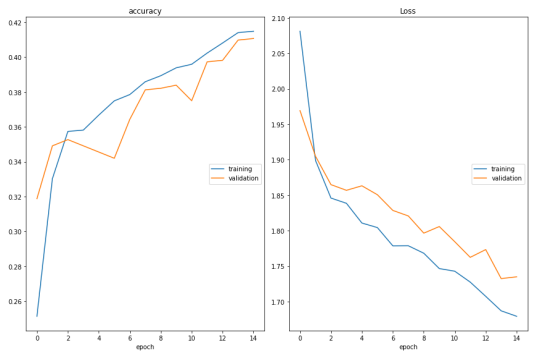

Figure 5 - Accuracy and Loss obtained for the train and validation datasets with Baseline Model 1.

CNN Baseline Model 2:

This model was created from the baseline model 1 and it was added four more Conv2D layers. Two of them with 128 filters and kernel size of (3,3) and the other two with 256 filters and a kernel size of (3,3). Between each of these groups, two Max Pooling layers were added with a pool size of (4,4) to reduce the spatial size of the representation to reduce the amount of parameters and consequently the computation in the network (reference [16]). After 15 epochs the model reached an accuracy of approximately 92% in the training data and an accuracy of 91% in the validation.

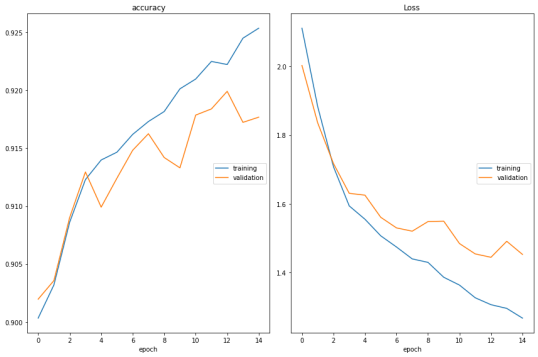

Figure 6 - Accuracy and Loss obtained for the train and validation datasets with Baseline Model 2.

CNN Baseline Model 3:

In the baseline model 3, the last two conv2D layers' filter was doubled (from 256 to 512) to increase learning and a batch normalization before each conv2D layers which means, 6 batch normalization layers. According to research, with batch normalization networks train faster and allow the usage of higher learning rates (reference [17]). By that, it was performed a run for 10 epochs and with an Adam leaning rate of 0.001. All the other parameters remained the same. After 10 epochs this model presented an accuracy of 62% in the training dataset and 53% in the validation.

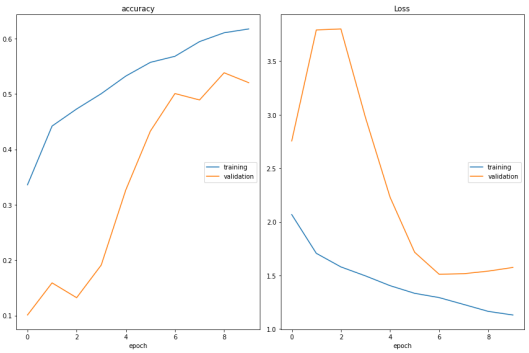

Figure 7 - Accuracy and Loss obtained for the train and validation datasets with Baseline Model 3.

4.3. Transfer learning

As previously mentioned, some of the most popular pre-trained methods for image classification were tested, namely the VGG16, ResNet50 and the Inception V3. The section below describes the applied models and obtained results.

VGG 16:

VGG 16 is a "16 layer transfer learning architecture", meaning it allows for the use of models trained on one problem as a starting point on a related problem.

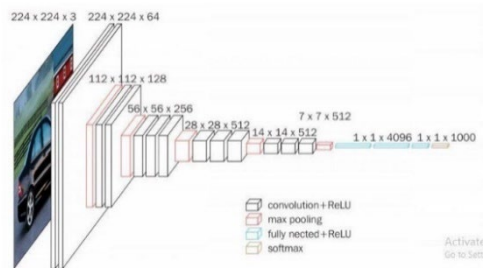

Figure 8 – VGG pretrained model illustration.

In the VGG16 based model, different operations and layers were applied namely, the flatten operation was defined, then two dense layers using the rectified linear activation function ("ReLU"), followed by a Dropout layer with a 20% probability, and lastly a dense layer with a softmax function is used to normalize the outputs, converting them from weighted sum values into probabilities that sum to one.

After 5 epochs this model presented an accuracy of 28,7% in the training dataset and 32% in the validation.

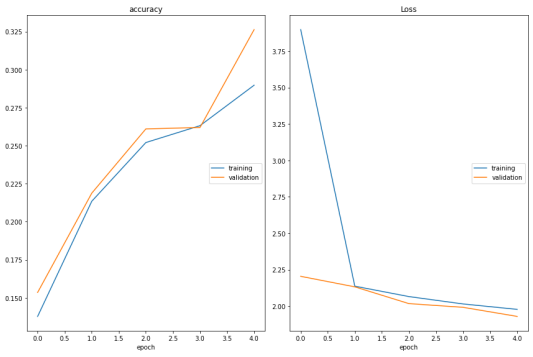

Figure 9 – Accuracy and Loss obtained for the train and validation datasets with VGG16.

Considering the identified constraints to train the VGG16, as it is a time and computational demanding process (i.e. it was not possible to train with more than 5 epochs considering limited computational resources), the project team decided to test two other models introduced a few years later, which are said to lead to boosts in accuracy and performance, i.e., ResNet50 and Inception V3.

ResNet50:

The **ResNet50** was introduced by a team in Microsoft in 2015. The model is a 50-layer convolutional neural network, that tries to overcome the vanishing gradient problem, using the concept of shortcut connections and a bottleneck structure which reduces the number of parameters and matrix multiplications enhancing performance (He, K. Zhang, X. Ren, S. Sun, J 2015).

The ResNet50 model was previously trained on ImageNet, which is an image database with thousands of images. Therefore, in this analysis the trained weights were considered, by downloading them from keras.

With the ResNet technique, other layers and conditions were tested, like a Global Average Pooling Layer, where idea is to generate one feature map for each corresponding category of the classification task, batch normalization technique to scale the activations, dropouts varying the probability parameter between 0.2 and 0.5 and lastly it was also considered a dense layer with softmax activation. With the ResNet technique, other layers and conditions were tested, like a Global Average Pooling Layer, where idea is to generate one feature map for each corresponding category of the classification task, batch normalization technique to scale the activations, dropouts varying the probability parameter between 0.2 and 0.5 and lastly it was also considered a dense layer with softmax activation.

After 10 epochs this model presented an accuracy of 23,6% in the training dataset and 25,7% in the validation.

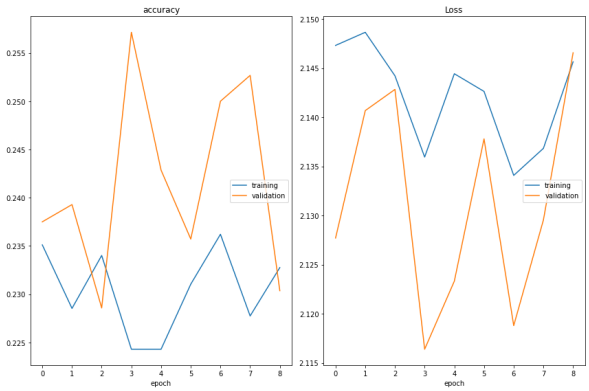

Figure 10 – Accuracy and Loss obtained for the train and validation datasets with ResNet50.

InceptionV3:

Regarding **InceptionV3**, the model was also introduced in 2015 and is an optimization of the model InceptionV1 created by Google. According to research, “the combination of lower parameter count and additional regularization with batch-normalized auxiliary classifiers and label-smoothing allows for training high quality networks on relatively modest sized training sets” (Szegedy et al., 2015).

In terms of model architecture, the model has 42 layers and it is constituted by: i) factorized convolutions and asymmetric convolutions, which allow to reduce the number of parameters involved in the neural network; ii) smaller convolutions, which allow to train the model faster; iii) auxiliary classifier, which acts as a regularizer and allows to improve the convergence of very deep neural networks; and, iv) efficient grid size reduction.

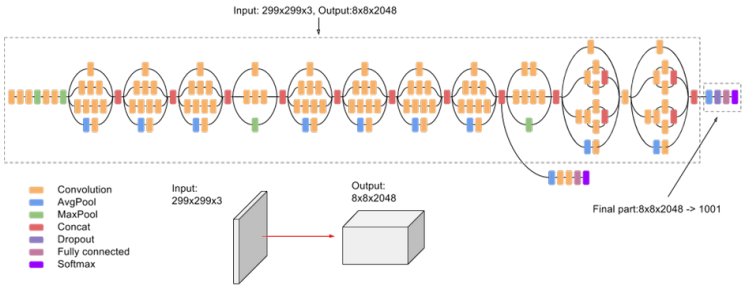

Figure 11 – High-level illustration of the Inception V3 model.

Similarly to the previous model (i.e. ResNet50), a pre-trained model was considered, and the weights of ImageNet were loaded. A Global Average Pooling layer was considered which will be applied to the output of the last convolutional block and images were resized to 229x229, as this is the standard for Inception V3.

Several combinations have been tested, however, the model that obtained the best result for accuracy with the Adams optimizer considered a Batch Normalization technique (applies a transformation to maintain the tensors' mean and standard deviation constant) with a momentum of 0,99 and an epsilon of 0,001. Followed by a dropout with 20% of probability, a dense layer with ReLu activation, another dropout with 20% of probability and a dense layer with softmax activation.

After 50 epochs this model presented an accuracy of 65,6% in the training dataset and 54,4% in the validation.

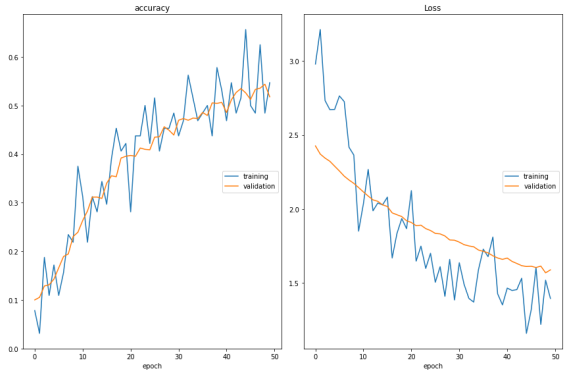

Figure 12 – Accuracy and Loss obtained for the train and validation datasets with InceptionV3.

5 Evaluation

To evaluate the developed models in unseen data, the models were applied to the test dataset. Thus, the different models presented the following best accuracies for the test dataset:

1. Baseline model 1 with an accuracy of 41%;
2. Baseline model 2 with an accuracy of 55%;

3. Baseline model 3 with an accuracy of 49,88%;
4. VGG16 with an accuracy of 23,88%;
5. ResNet50 with an accuracy of 10,4%; and,
6. InceptionV3 with an accuracy of 65,5%.

6 Conclusion

The goal of this work was to recognize a meal from an image, i.e. to predict the class of a specific meal in order to apply the correct price to be requested to a specific customer of Cibus. After developing and testing several approaches, the model that presented the best results to classify the 10 classes of food defined was the Inception V3, with an achieved accuracy of 65,5% in the test dataset.

7 References

The group has considered both published research and other useful documentation published online, namely:

1. Gupta, A. (2021). A Comprehensive Guide on Deep Learning Optimizers.
2. Aremu, T. (2022). Impact of Optimizers in Image Classifiers. Available at: GitHub - Ti-Oluwanimi/Impact-of-Optimizers: A practical outlook on the most popular optimizers used in deep learning.
3. Kandel, I. Castelli, M. Popovič, A. (2020) Comparative Study of First Order Optimizers for Image Classification Using Convolutional Neural Networks on Histopathology Images. Available at: https://www.mdpi.com/2313-433X/6/9/92.
4. Li, C. Kaan Tokgoz, K. Fukawa, M, and others (2021). Data Augmentation for Inertial Sensor Data in CNNs for Cattle Behavior Classification. Available at: https://ieeexplore.ieee.org/abstract/document/9566833.
5. He, K. Zhang, X. Ren, S. Sun, J (2015). Deep Residual Learning for Image Recognition. Available at: https://arxiv.org/abs/1512.03385.
6. Szegedy, C., Vanhoucke, V. , Ioffe, S., Shlens, J. and Wojna, Z. (2015). Rethinking the Inception Architecture for Computer Vision. Available at: https://arxiv.org/abs/1512.00567.
7. Tensorflow documentation. Available at the following link: https://www.tensorflow.org/api_docs/python/tf/config/run_functions_eagerly
8. Keras documentation regarding callbacks. Available at the following link: https://keras.io/api/callbacks/.

9. Keras documentation regarding *ModelCheckpoint*. Available at the following link: https://keras.io/api/callbacks/model_checkpoint/.

10. Information regarding the package *Livelossplot*. Available at the following link: https://p.migdal.pl/livelossplot/.

11. Information regarding Transfer Learning, including python code. Available at the following link: www.learndatasci.com/tutorials/hands-on-transfer-learning-keras/.

12. Tutorial on using Keras flow_from_directory and generators. Available at the following: https://vijayabhaskar96.medium.com/tutorial-image-classification-with-keras-flow-from-directory-and-generators-95f75ebe5720.

13. Documentation regarding VGG16. Available at the following: https://datagen.tech/guides/computer-vision/vgg16/.

14. Documentation regarding Inception V3. Available at the following: https://cloud.google.com/tpu/docs/inception-v3-advanced.

15. Documentation regarding Inception V3 and ResNet50. Available at the following: https://blog.paperspace.com/popular-deep-learning-architectures-resnet-inceptionv3-squeezenet/.

16. Documentation regarding Max Pooling. Available at the following: https://deepai.org/machine-learning-glossary-and-terms/max-pooling

17. Documentation regarding Batch Normalization. Available at the following: https://towardsdatascience.com/batch-normalization-8a2e585775c9