



Msc Computer Engineering  
Intelligent Systems

# Food Recogniser

Lorenzo Bataloni  
Edoardo Pantè  
Claudio Daka

Accademic Year 22/23

<b>1.Introduction</b>	<b>4</b>
<b>2.Dataset</b>	<b>4</b>
2.1. Dataset Structure	4
2.2. Dataset Elaboration	7
<b>3.Model From Scratch</b>	<b>7</b>
3.1. First Model	7
3.1.3. Result	10
3.2. Second Model	11
3.2.2. Result	14
3.2.3. Model without data augmentation	15
3.3. Third Model	16
3.3.2. Results	19
3.4. Tuned Third Model	21
<b>4.Pre-trained Networks</b>	<b>22</b>
4.1. Introduction	22
4.2. Feature Extraction	22
4.3. Pre-processing	22
4.4. MobileNetV2	23
4.4.1. Experiment 0 - Base model	23
4.4.2. Tuned base model	26
4.4.3. Experiment 1 - Tuned freezed model	28
4.4.4. Fine tuning	30
4.4.4. Experiment 2 - Unfreeze 1	31
4.4.5. Experiment 3 - Unfreeze 2	33
4.4.6. Experiment 4 - Unfreeze 3	35
4.5. InceptionV3	37
4.5.1 Experiment 0 - Base model	37
4.5.2. Experiment 1 - Unfreeze 1	39
4.5.3. Experiment 2 - Unfreeze 2	41
4.5.4. Experiment 3 - Unfreeze 3	43

# 1. Introduction

Food Recognition is an image recognition system that can recognize and identify up to 101 different categories of food. The use of this type of system is very broad and can range from use in apps for example for diet or meal management to use to help people with visual impairments for example interpret menus or otherwise overcome any barriers.

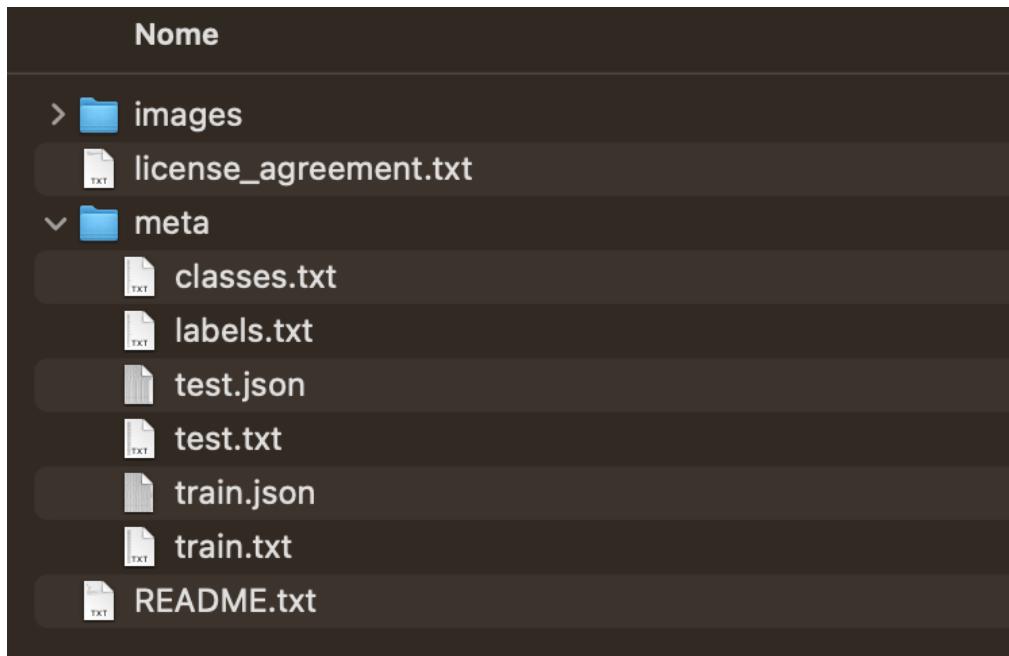
To carry out this task, different technologies were used to also make comparisons to determine which system offered the best performance. A CNN was created from scratch using tuning techniques to improve performance, two pre-trained CNNs were then used using fine tuning techniques. Of these, one was trained using images similar to those in the dataset used by us.

## 2. Dataset

The dataset used is the food-101 dataset. A dataset containing images of food, these images belong to 101 different classes. For each class there are 1000 images of different sizes, for a total of about 5 GB of dataset.

### 2.1. Dataset Structure

Unlike standard datasets, this dataset does not have a ready-to-use structure, and a scripting step was required to process the images and create a suitable structure. As we can see in the image, the dataset contains a single images folder, with a folder for each class inside, and within these are all the images, both test and train images.



In addition to the images folder we have the meta folder that contains files that contain information about the dataset. Specifically, the files test.json and train.json contain the names of the files to be used for test and train, respectively.

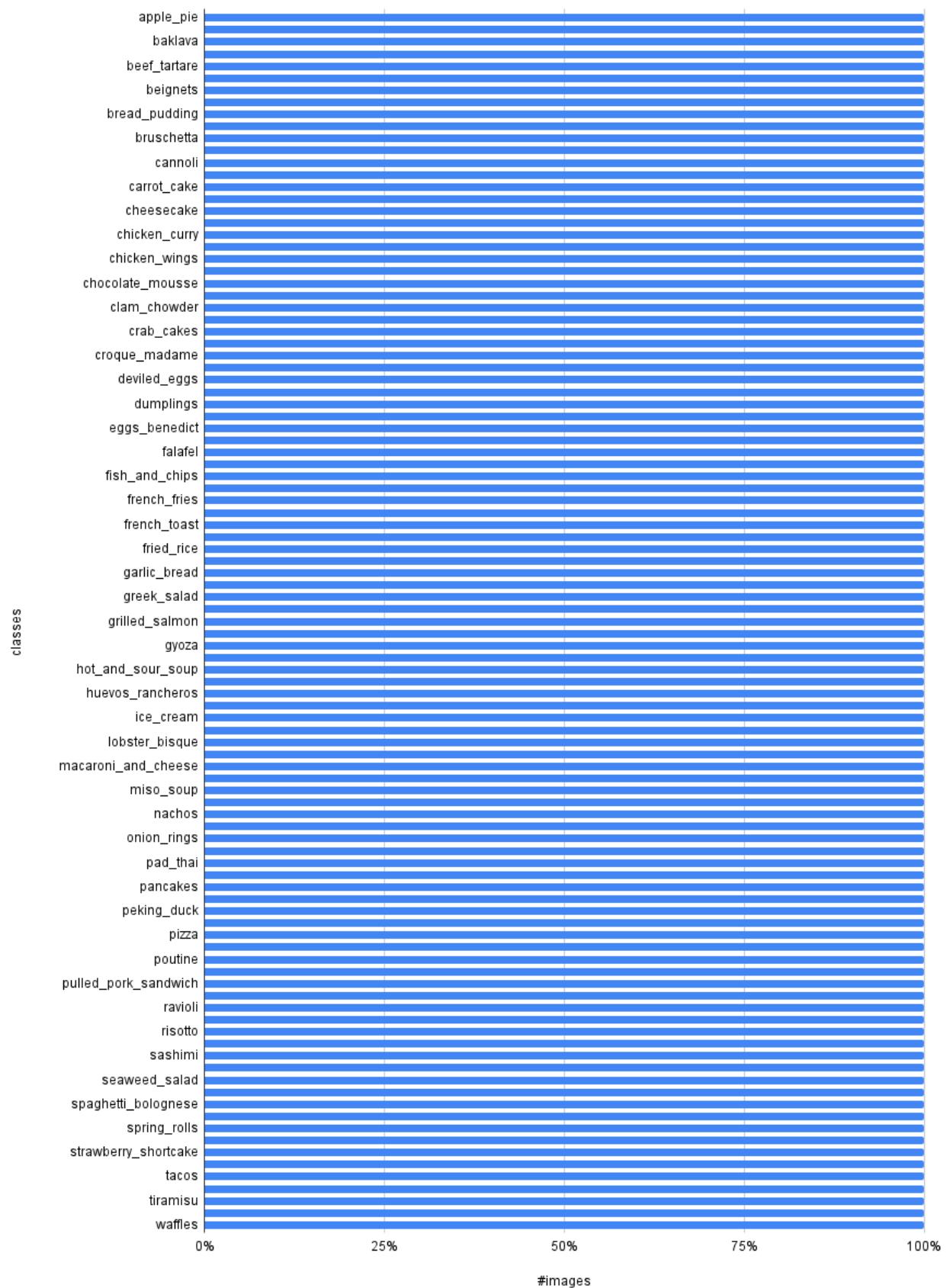
A script was made that would go to read these files and extract the files so that we would have a structure suitable for use with our networks. In addition, a portion of the images for the train is used for validation.

The final structure of the dataset is as follows, with three different directories for train, test, and validation. Each directory within it has 101 directories, one per class, containing the relevant images.



As we can see from the graph below, the dataset is perfectly balanced.

## #images per class



## 2.2. Dataset Elaboration

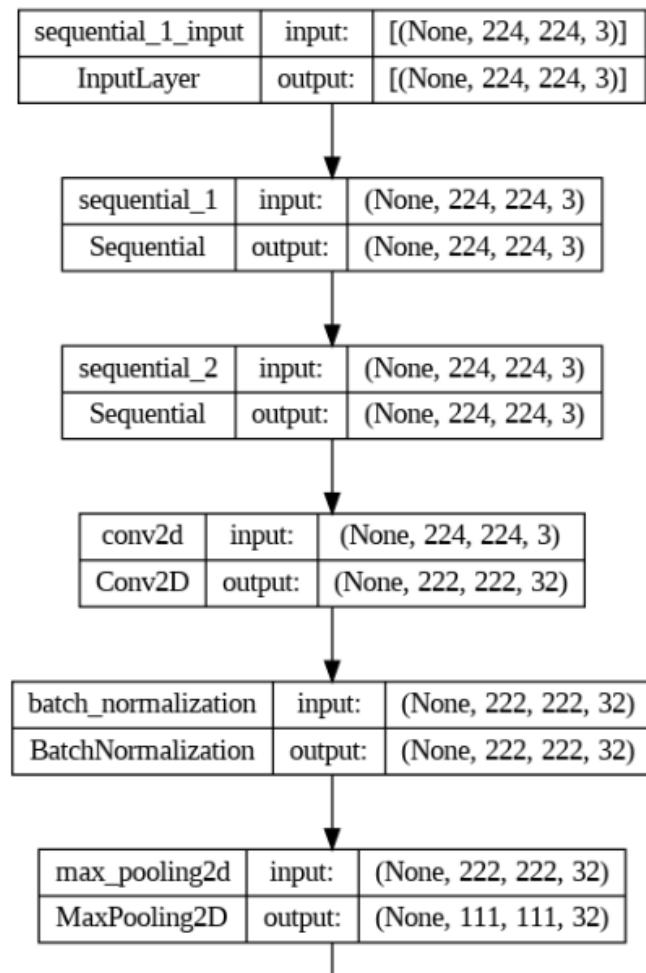
Following processing, initial tests were carried out with both the from scratch and pre-trained models. Problems were immediately encountered due to the size of the dataset and the number of classes. A major problem was the tool used, namely google colab. This tool in fact has limitations for use, especially with regard to the GPU. It was therefore decided to go for thinning the images and use a few hundred images rather than a thousand per class so as not to have any impediments.

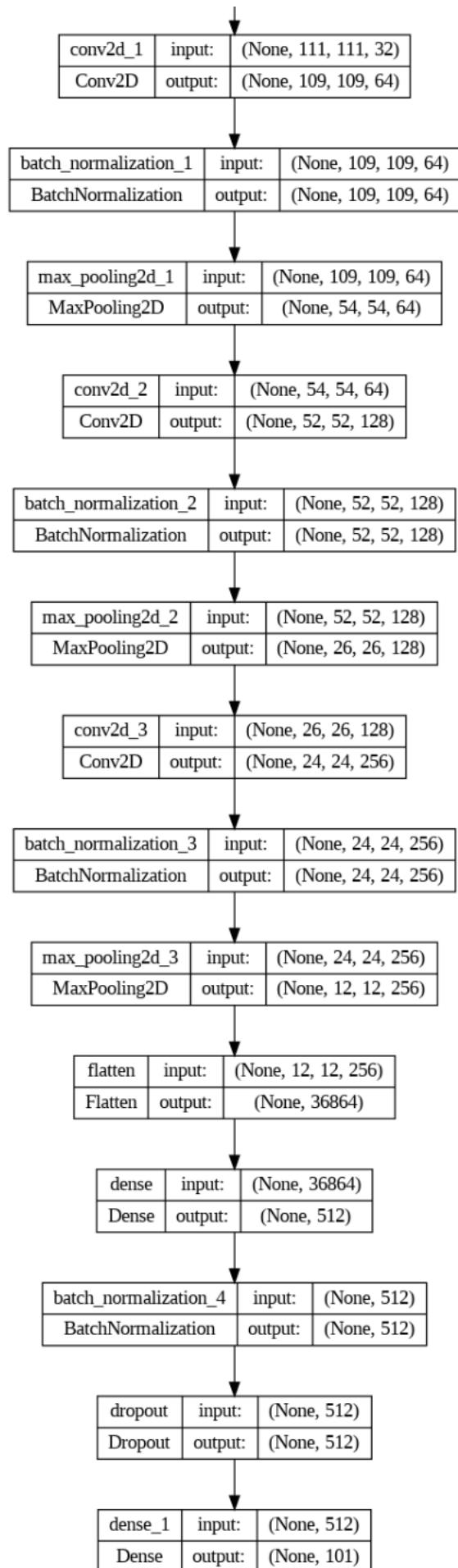
## 3. Model From Scratch

For the from scratch models, we started with simple networks and then went on to use more complex models to counteract any overfitting.

### 3.1. First Model

The first model is the simplest one we used; it has 4 convolutional blocks starting from 32 filters up to 256. All blocks are separated by batch normalization and max pooling layers. We ran a train with 25 epochs with a learning rate ranging from 0.1 up to 1e-5.

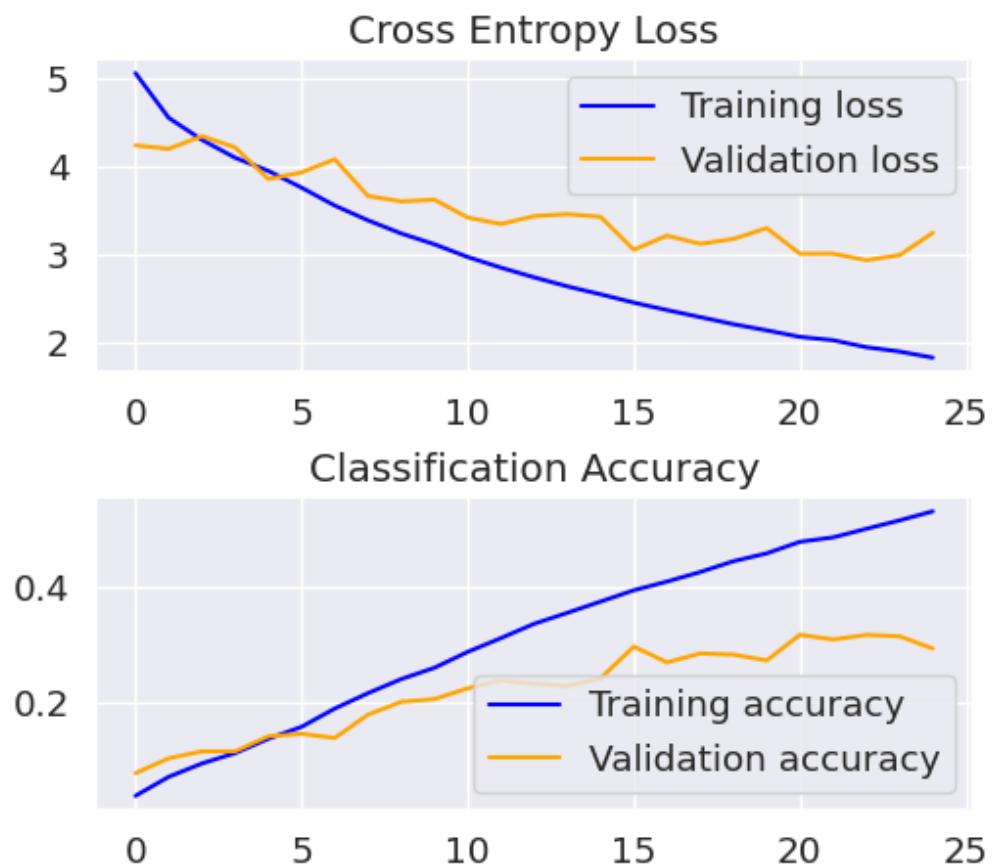


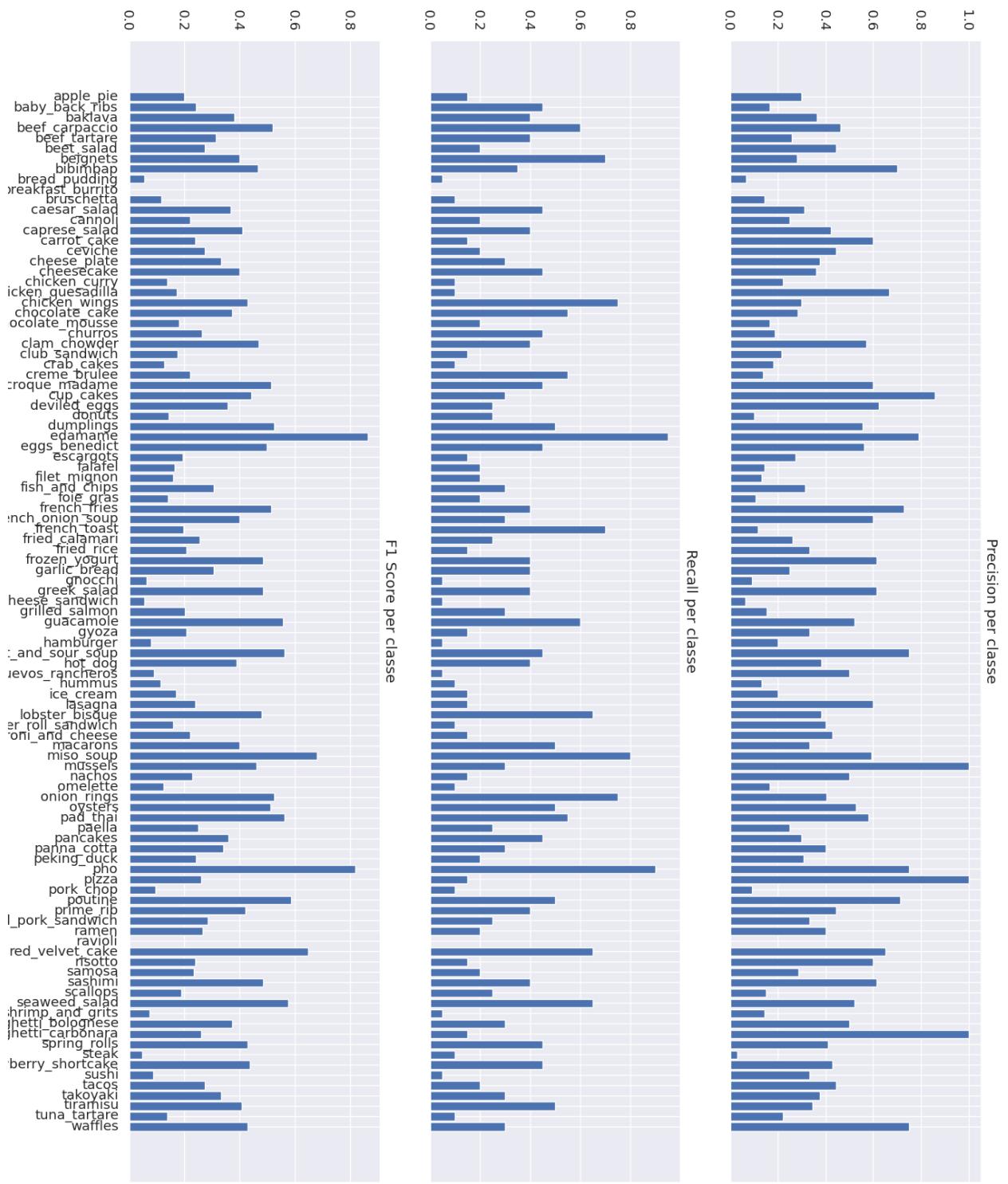


Layer (type)	Output Shape	Param #
sequential_1 (Sequential)	(None, 224, 224, 3)	0
sequential_2 (Sequential)	(None, 224, 224, 3)	0
conv2d (Conv2D)	(None, 222, 222, 32)	896
batch_normalization (BatchNormalization)	(None, 222, 222, 32)	128
max_pooling2d (MaxPooling2D)	(None, 111, 111, 32)	0
conv2d_1 (Conv2D)	(None, 109, 109, 64)	18496
batch_normalization_1 (BatchNormalization)	(None, 109, 109, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 54, 54, 64)	0
conv2d_2 (Conv2D)	(None, 52, 52, 128)	73856
conv2d_3 (Conv2D)	(None, 50, 50, 128)	147584
batch_normalization_2 (BatchNormalization)	(None, 50, 50, 128)	512
max_pooling2d_2 (MaxPooling2D)	(None, 25, 25, 128)	0
dropout (Dropout)	(None, 25, 25, 128)	0
conv2d_4 (Conv2D)	(None, 23, 23, 256)	295168
conv2d_5 (Conv2D)	(None, 21, 21, 256)	590080
batch_normalization_3 (BatchNormalization)	(None, 21, 21, 256)	1024
max_pooling2d_3 (MaxPooling2D)	(None, 10, 10, 256)	0
dropout_1 (Dropout)	(None, 10, 10, 256)	0
flatten (Flatten)	(None, 25600)	0
dense (Dense)	(None, 512)	13107712
batch_normalization_4 (BatchNormalization)	(None, 512)	2048
dropout_2 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 101)	51813
<hr/>		
Total params: 14,289,573		
Trainable params: 14,287,589		
Non-trainable params: 1,984		

### 3.1.3. Result

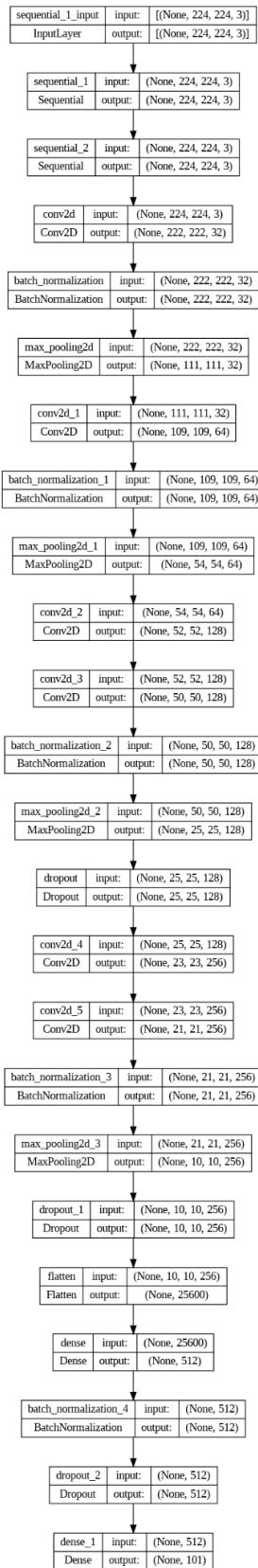
Despite the presence of the data augmentation and dropout layer, as we can see from the graphs, overfitting is present. In fact, the curve related to validation goes to a plateau after a certain number of epochs and tends to stabilize. After training the accuracy of the network settles around 29% with the test dataset.





### 3.2. Second Model

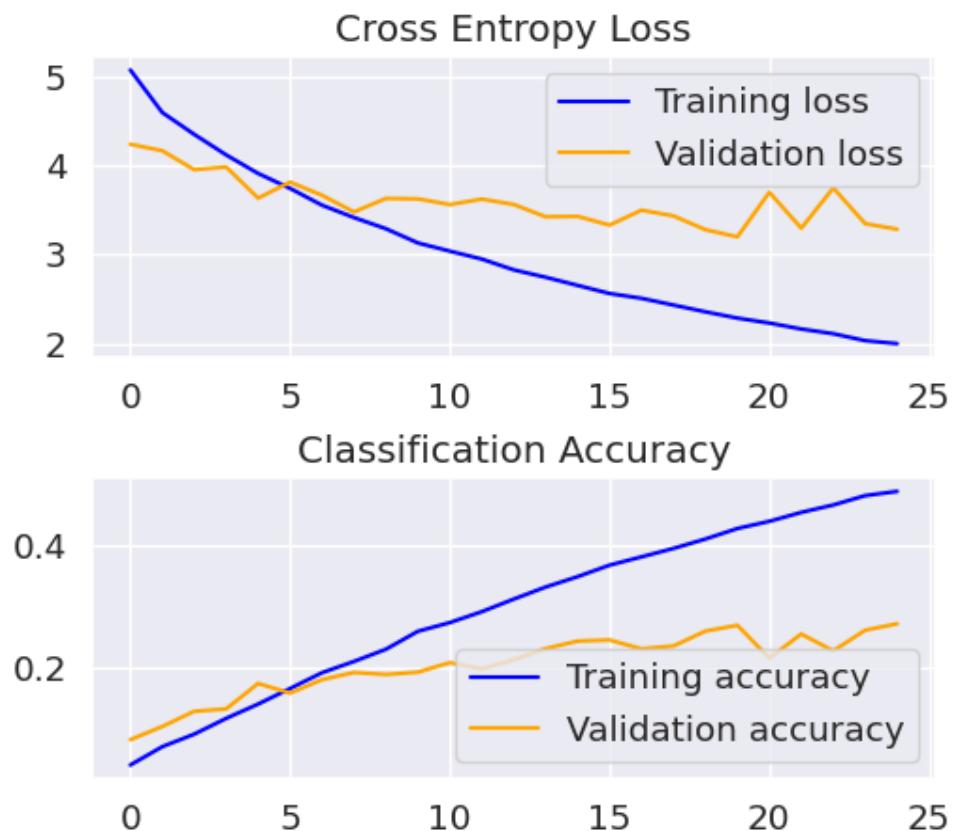
For the second model, we modified the convolutional layers by adding a convolutional layer to the last 2 layers to try to increase the accuracy of the model. We then added dropout layers to try to decrease overfitting.

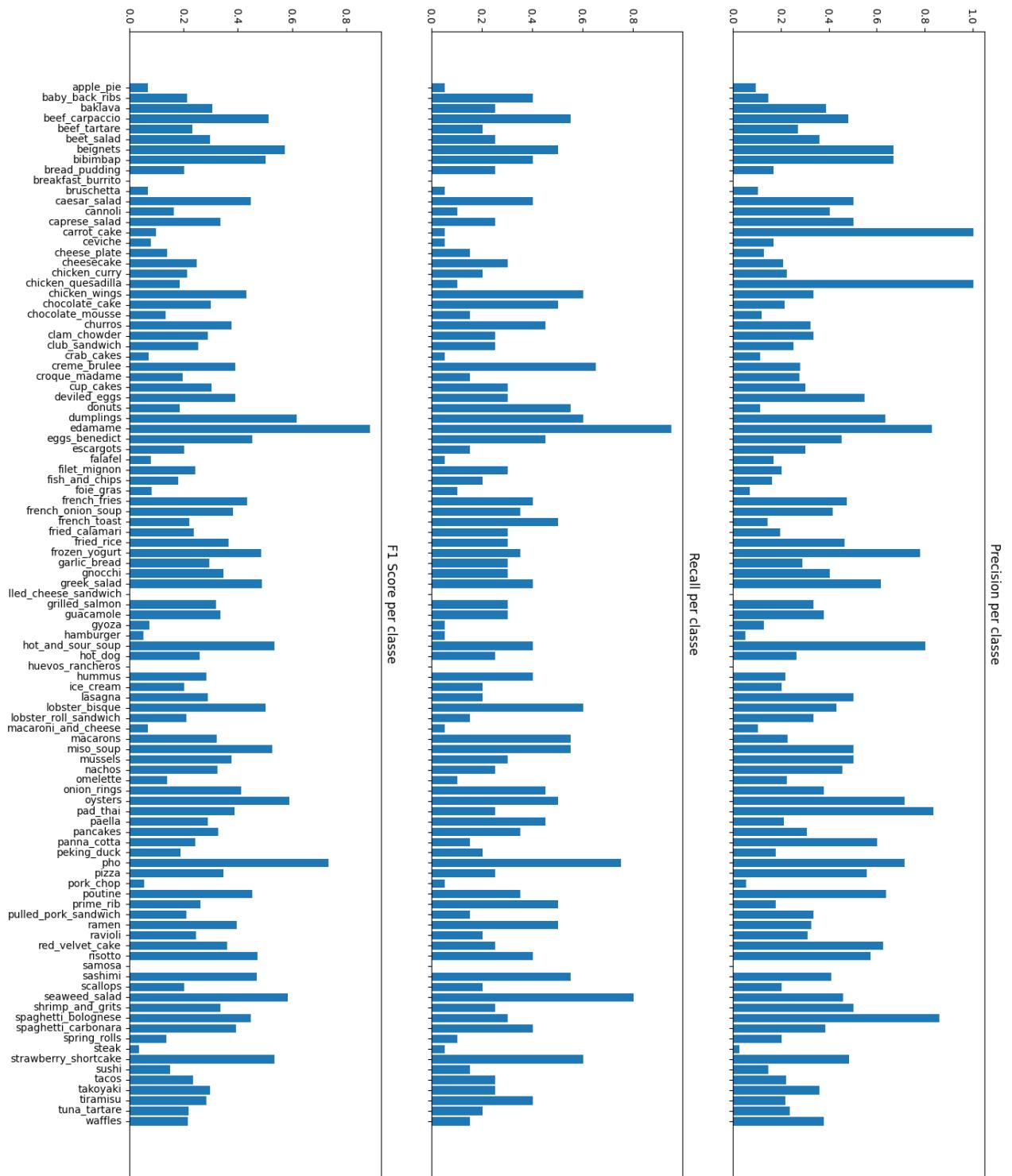


Layer (type)	Output Shape	Param #
sequential_1 (Sequential)	(None, 224, 224, 3)	0
sequential_2 (Sequential)	(None, 224, 224, 3)	0
conv2d (Conv2D)	(None, 222, 222, 32)	896
batch_normalization (BatchN ormalization)	(None, 222, 222, 32)	128
max_pooling2d (MaxPooling2D )	(None, 111, 111, 32)	0
conv2d_1 (Conv2D)	(None, 109, 109, 64)	18496
batch_normalization_1 (Batch hNormalization)	(None, 109, 109, 64)	256
max_pooling2d_1 (MaxPooling 2D)	(None, 54, 54, 64)	0
conv2d_2 (Conv2D)	(None, 52, 52, 128)	73856
conv2d_3 (Conv2D)	(None, 50, 50, 128)	147584
batch_normalization_2 (Batch hNormalization)	(None, 50, 50, 128)	512
max_pooling2d_2 (MaxPooling 2D)	(None, 25, 25, 128)	0
dropout (Dropout)	(None, 25, 25, 128)	0
conv2d_4 (Conv2D)	(None, 23, 23, 256)	295168
conv2d_5 (Conv2D)	(None, 21, 21, 256)	590080
batch_normalization_3 (Batch hNormalization)	(None, 21, 21, 256)	1024
max_pooling2d_3 (MaxPooling 2D)	(None, 10, 10, 256)	0
dropout_1 (Dropout)	(None, 10, 10, 256)	0
flatten (Flatten)	(None, 25600)	0
dense (Dense)	(None, 512)	13107712
batch_normalization_4 (Batch hNormalization)	(None, 512)	2048
dropout_2 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 101)	51813
<hr/>		
Total params:	14,289,573	
Trainable params:	14,287,589	
Non-trainable params:	1,984	

### 3.2.2. Result

As we expected, the model achieves a better accuracy of around 35%. However, efforts to reduce overfitting are not enough. In fact, as we can see from the graphs, again the curves related to validation go to settling and again reach a plateau.





### 3.2.3. Model without data augmentation

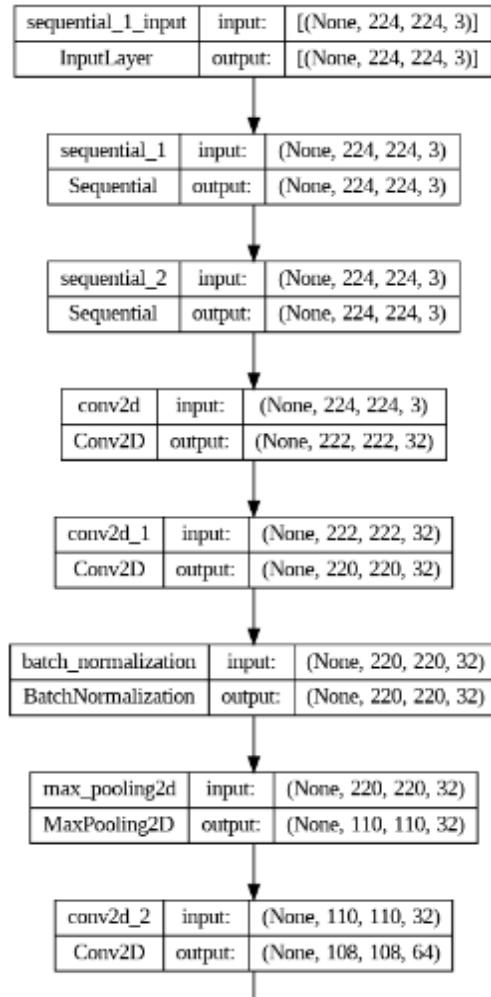
Surprised by the difficulty of combating this overfitting, we tried running a new train by removing the data augmentation layer. The result reassured us, in fact, already from the first epochs the difference between the train and validation curves is very marked and far exceeds the difference found in previous experiments. This indicates good progress in

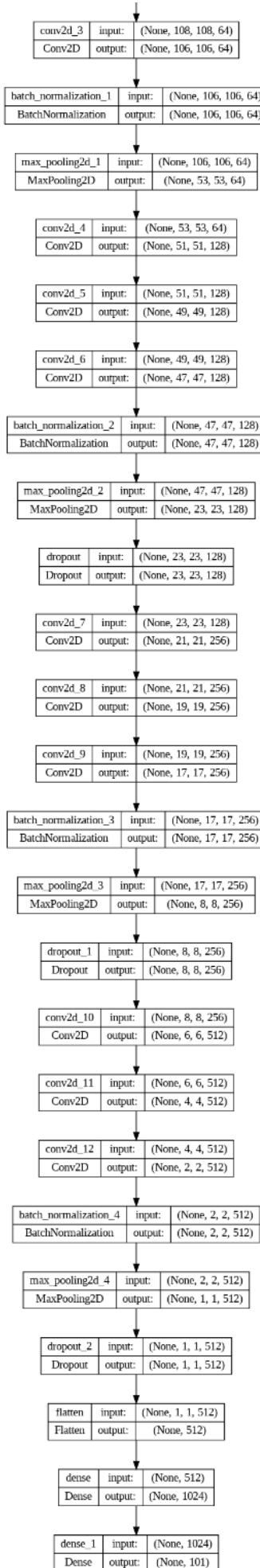
combating overfitting despite the fact that it remained even in the experiments already completed.

```
757/757 [=====] - 183s 242ms/step - loss: 1.2929 - accuracy: 0.7013 - val_loss: 4.2040 - val_accuracy: 0.1463
Epoch 7/25
757/757 [=====] - ETA: 0s - loss: 0.6179 - accuracy: 0.8878
Epoch 7: val_accuracy did not improve from 0.14633
757/757 [=====] - 182s 241ms/step - loss: 0.6179 - accuracy: 0.8878 - val_loss: 4.1720 - val_accuracy: 0.1419
Epoch 8/25
757/757 [=====] - ETA: 0s - loss: 0.3018 - accuracy: 0.9599
Epoch 8: val_accuracy did not improve from 0.14633
757/757 [=====] - 181s 239ms/step - loss: 0.3018 - accuracy: 0.9599 - val_loss: 4.3199 - val_accuracy: 0.1374
Epoch 9/25
757/757 [=====] - ETA: 0s - loss: 0.1641 - accuracy: 0.9840
Epoch 9: val_accuracy did not improve from 0.14633
757/757 [=====] - 181s 239ms/step - loss: 0.1641 - accuracy: 0.9840 - val_loss: 4.3072 - val_accuracy: 0.1434
Epoch 10/25
757/757 [=====] - ETA: 0s - loss: 0.1077 - accuracy: 0.9906
Epoch 10: val_accuracy did not improve from 0.14633
757/757 [=====] - 159s 210ms/step - loss: 0.1077 - accuracy: 0.9906 - val_loss: 4.3982 - val_accuracy: 0.1347
Epoch 11/25
757/757 [=====] - ETA: 0s - loss: 0.0872 - accuracy: 0.9923
Epoch 11: val_accuracy did not improve from 0.14633
757/757 [=====] - 162s 214ms/step - loss: 0.0872 - accuracy: 0.9923 - val_loss: 4.5704 - val_accuracy: 0.1262
Epoch 12/25
757/757 [=====] - ETA: 0s - loss: 0.0868 - accuracy: 0.9918
Epoch 12: val_accuracy did not improve from 0.14633
757/757 [=====] - 161s 213ms/step - loss: 0.0868 - accuracy: 0.9918 - val_loss: 4.7791 - val_accuracy: 0.1240
Epoch 13/25
```

### 3.3. Third Model

To try to increase performance, we added complexity to the model and added additional dropout levels. We added new convolutional layers with additional filters. We also tried increasing epochs but as we will see it did not help.

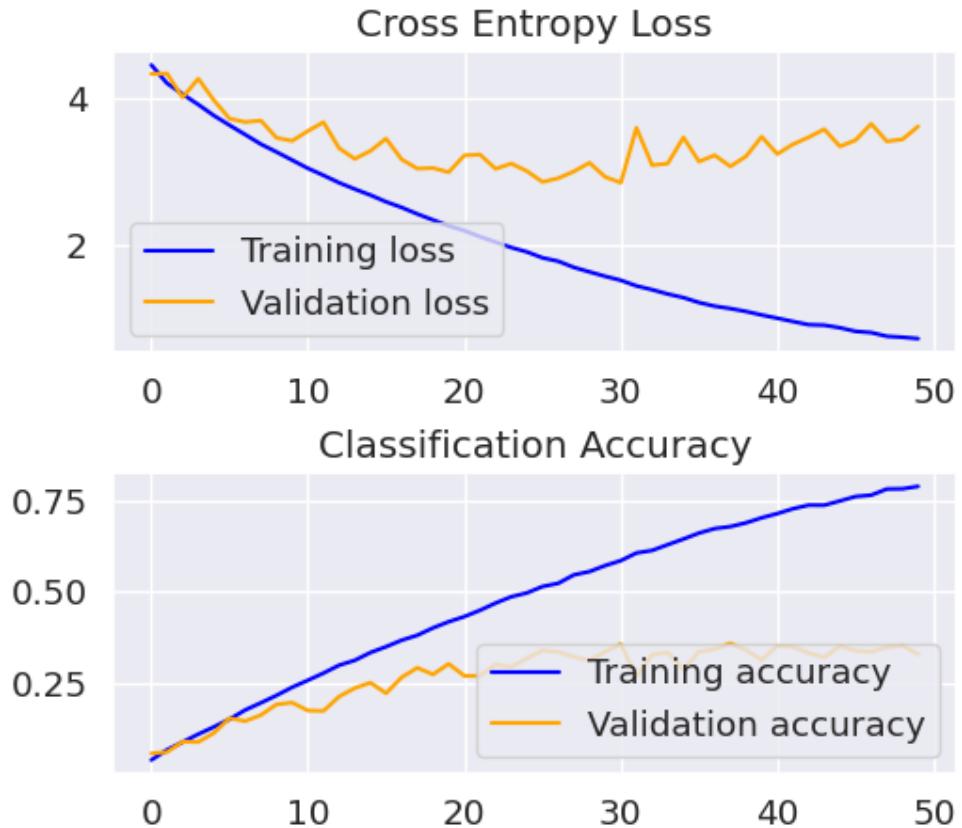


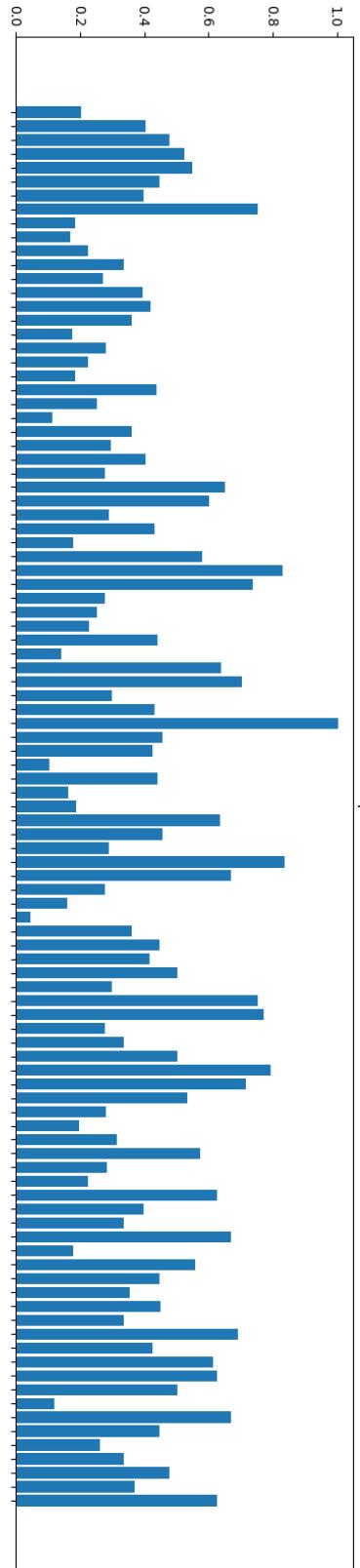
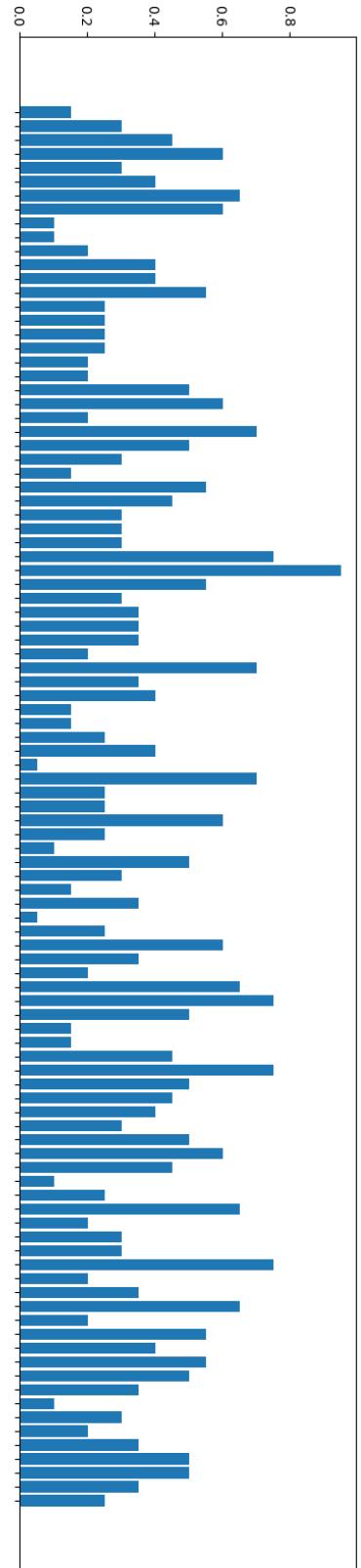
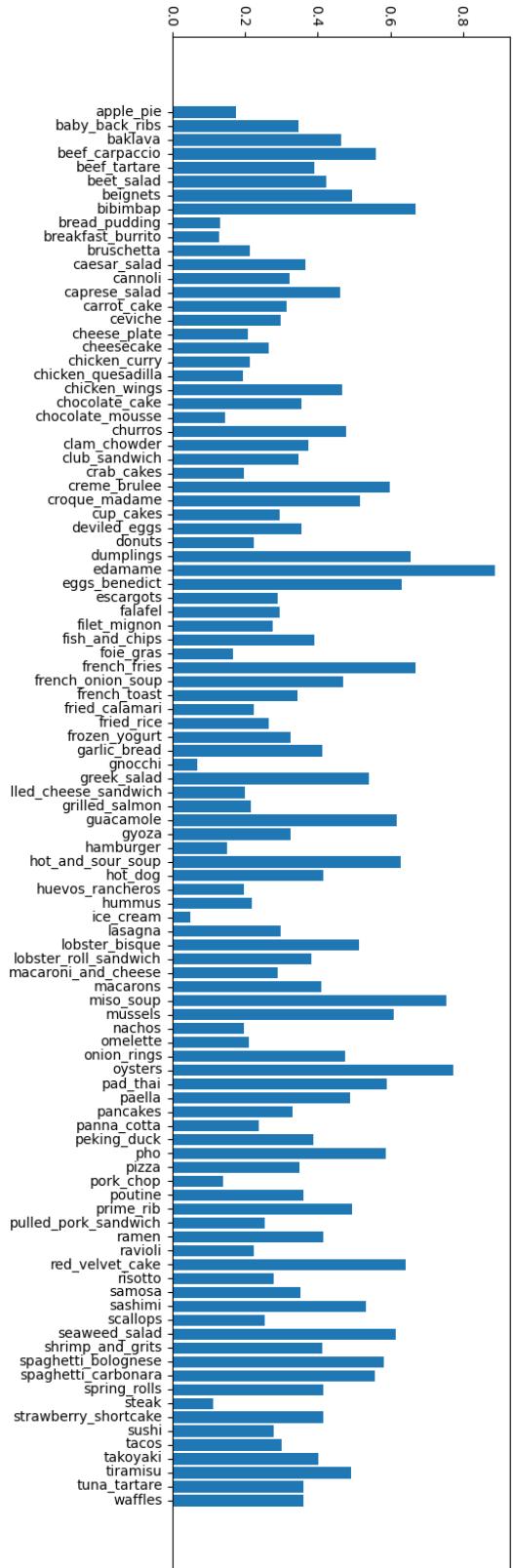


Layer (type)	Output Shape	Param #
sequential_1 (Sequential)	(None, 224, 224, 3)	0
sequential_2 (Sequential)	(None, 224, 224, 3)	0
conv2d (Conv2D)	(None, 222, 222, 32)	896
batch_normalization (BatchN ormalization)	(None, 222, 222, 32)	128
max_pooling2d (MaxPooling2D )	(None, 111, 111, 32)	0
conv2d_1 (Conv2D)	(None, 109, 109, 64)	18496
batch_normalization_1 (Batch hNormalization)	(None, 109, 109, 64)	256
max_pooling2d_1 (MaxPooling 2D)	(None, 54, 54, 64)	0
conv2d_2 (Conv2D)	(None, 52, 52, 128)	73856
conv2d_3 (Conv2D)	(None, 50, 50, 128)	147584
batch_normalization_2 (Batch hNormalization)	(None, 50, 50, 128)	512
max_pooling2d_2 (MaxPooling 2D)	(None, 25, 25, 128)	0
dropout (Dropout)	(None, 25, 25, 128)	0
conv2d_4 (Conv2D)	(None, 23, 23, 256)	295168
conv2d_5 (Conv2D)	(None, 21, 21, 256)	590080
batch_normalization_3 (Batch hNormalization)	(None, 21, 21, 256)	1024
max_pooling2d_3 (MaxPooling 2D)	(None, 10, 10, 256)	0
dropout_1 (Dropout)	(None, 10, 10, 256)	0
flatten (Flatten)	(None, 25600)	0
dense (Dense)	(None, 512)	13107712
batch_normalization_4 (Batch hNormalization)	(None, 512)	2048
dropout_2 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 101)	51813
<hr/>		
Total params: 14,289,573		
Trainable params: 14,287,589		
Non-trainable params: 1,984		

### 3.3.2. Results

This third model, given the larger number of filters, allows accuracy close to 40 percent. However as we see from the graphs there remains a large component of overfitting despite efforts to try to lower it.





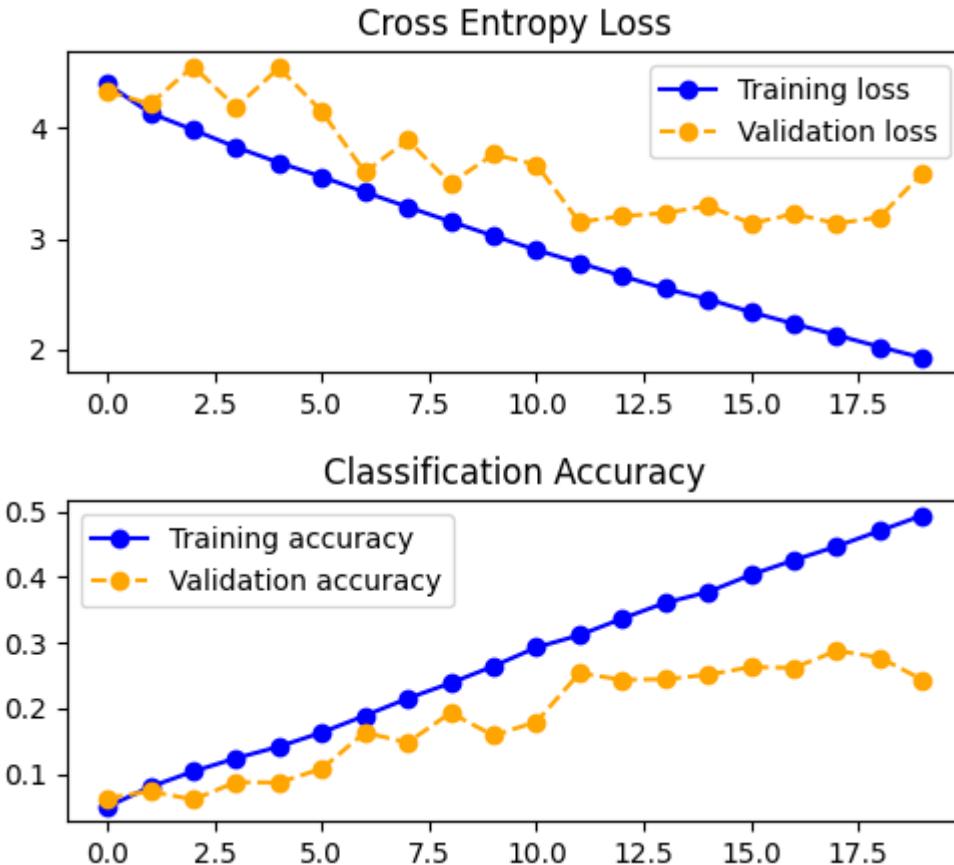
F1 Score per classe

Recall per classe

Precision per classe

### 3.4. Tuned Third Model

We then finally tried tuning the third model to see the differences in performance. We used fewer epochs given the previous performance and obtained a better result from the overfitting point of view.



As we can see from the graphs, the phenomenon of overfitting remains present but the gap between the curves remains controlled, despite the fact that a plateau tends to occur here as well.

## 4. Pre-trained Networks

### 4.1. Introduction

This section describes the results obtained using two different pretrained models (MobileNet and InceptionV3) that were pretrained on other large-scale image datasets.

These networks have been widely used for tasks such as image classification, semantic segmentation, and real-time image processing on mobile devices.

We have taken these pretrained models from the tensorflow.keras.application module and we choose the models with the weights obtained during training with the ImageNet dataset.

We choose an input of (224,224,3).

For each experiment, we use the Adam optimizer and the categorical cross-entropy as loss function.

For each network we computed some of the most important metrics: precision, recall and f1 score.

Due to the fact that we are dealing with 101 classes it is impossible to represent the confusion matrix in an understandable way.

### 4.2. Feature Extraction

Feature extraction consists of using the representations learned by a previous network to extract interesting features from new samples. The convolutional bases of the MobileNetV2 and InceptionV3 are frozen and the features that they extract are then sent to new classifiers, which are trained from scratch.

### 4.3. Pre-processing

Pre-processing the dataset is an important step before training a deep learning model like MobileNetV2 or InceptionV3. Here are the pre-processing steps we used for these pre-trained networks:

Resize Images: these networks expect images of a specific size as input. Resize all the images in the dataset to a consistent size, such as 224x224 pixels, to match the input size expected by the networks. Resizing ensures that all images have the same dimensions.

Normalization: Convert the resized images into a floating-point representation, typically in the range of 0 to 1. Divide the pixel values by 255, which is the maximum pixel value for an 8-bit image representation, to normalize the values between 0 and 1.

Data Augmentation: Augmenting the dataset with additional variations of the images can help improve the model's generalization ability. We have applied techniques such as random

rotations, flips, and zooms to generate new training samples from existing ones. Data augmentation helps models to learn robust features and reduces the risk of overfitting.

**Splitting the Dataset:** Divide the dataset into training, validation, and testing subsets. We have used 300 images per class and we have divided each set of images in this way:

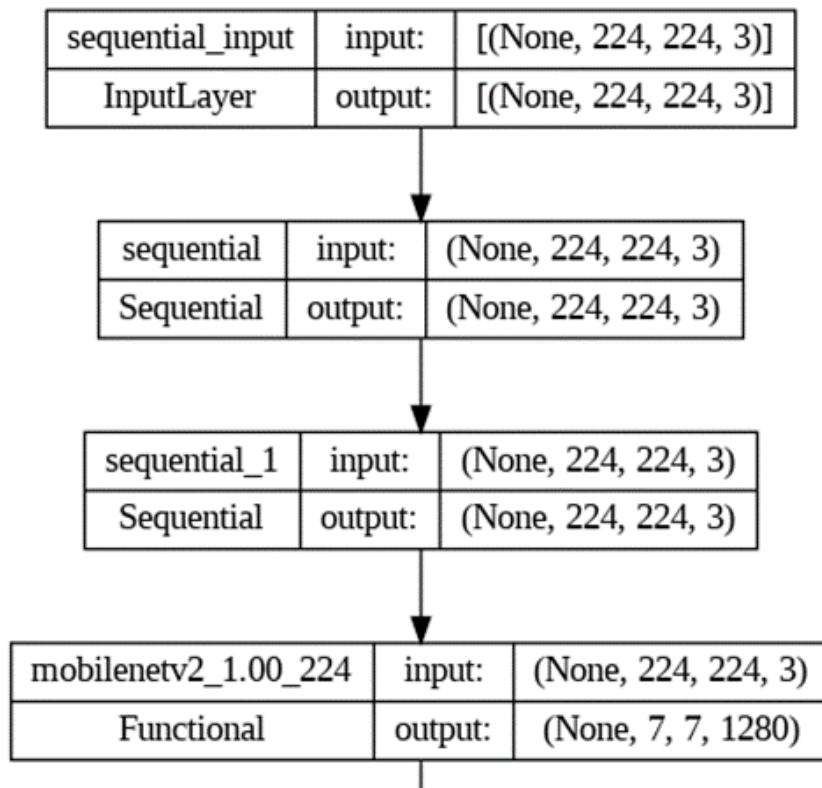
- Training : 240 images
- Validation: 40 images
- Test: 20 images

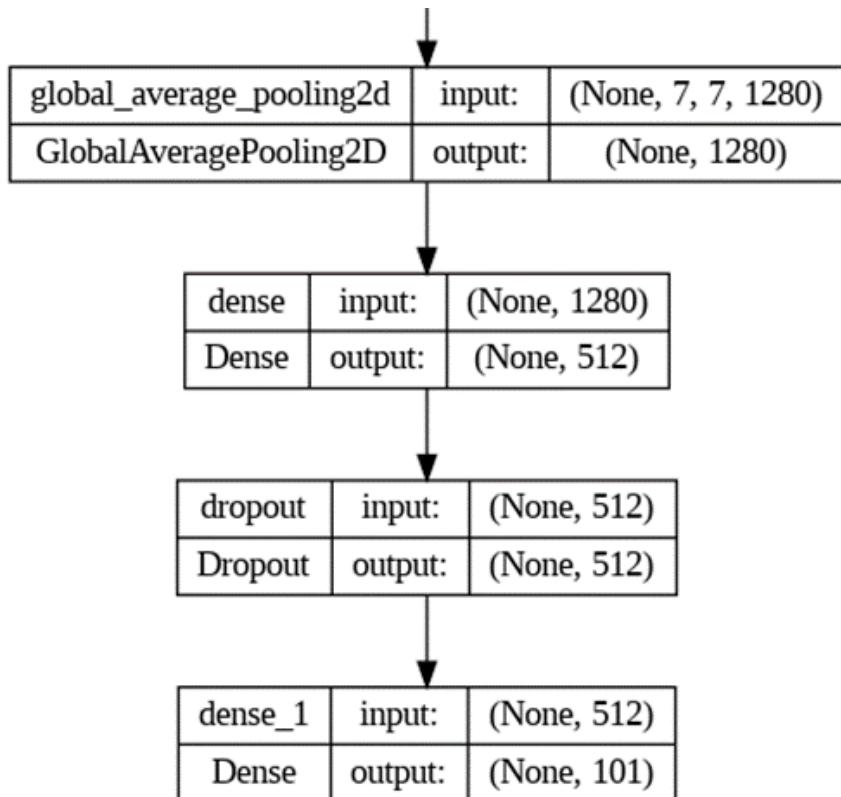
## 4.4. MobileNetV2

MobileNetV2 is a convolutional neural network architecture proposed for mobile and embedded vision applications. Designed for image processing on mobile devices. It is an improved and optimized version of MobileNetV1, developed by Google.

### 4.4.1. Experiment 0 - Base model

The MobileNetV2 has been taken as base model (with frozen weights) and on top of it has been built a GlobalAveragePooling2D layer, a dense layer, a dropout layer and an output layer. MobileNetV2 uses a Global Average Pooling layer to reduce the spatial dimensions of the feature map obtained from the convolutional layers, while retaining the important information. It computes the average of all activation values across all spatial dimensions and contains information about the spatial hierarchy. We could use a Flatten layer instead, but it may not be optimal because it simply takes the output of the previous layer and flattens it into a single long vector, losing all spatial information.

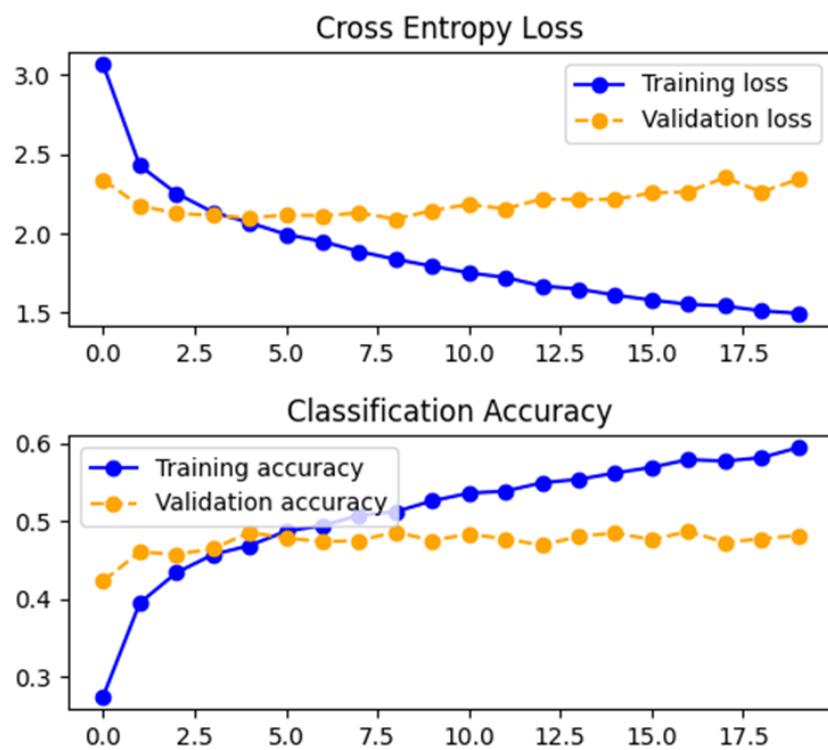




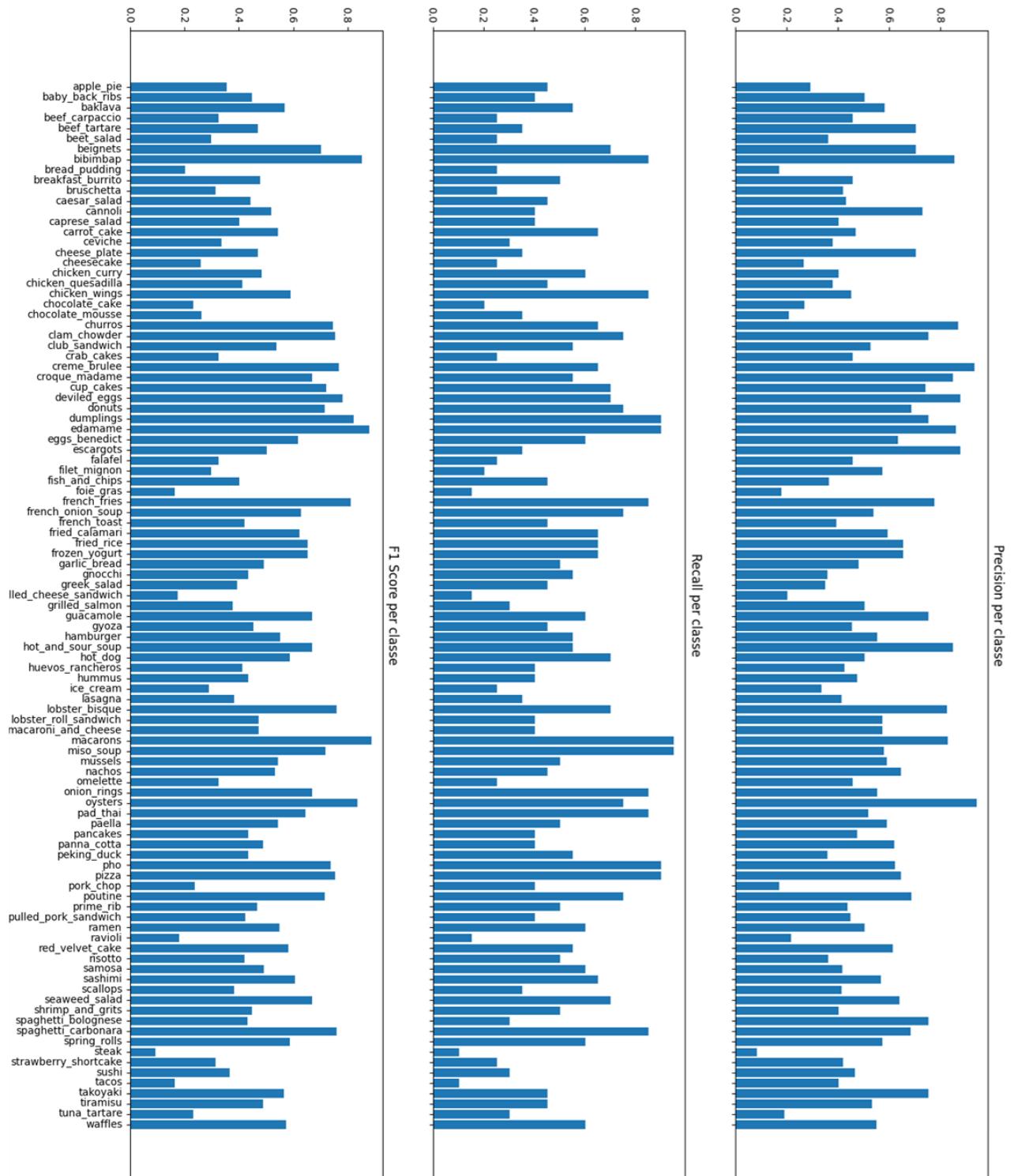
**Total params:** 2,965,669

**Trainable params:** 707,685

**Non-trainable params:** 2,257,984



After 5 epochs, we observe overfitting: the accuracy on the training set is larger than the one on the validation set and the accuracy on the validation set does not increase significantly anymore, which means that the network starts to memorize the training data.

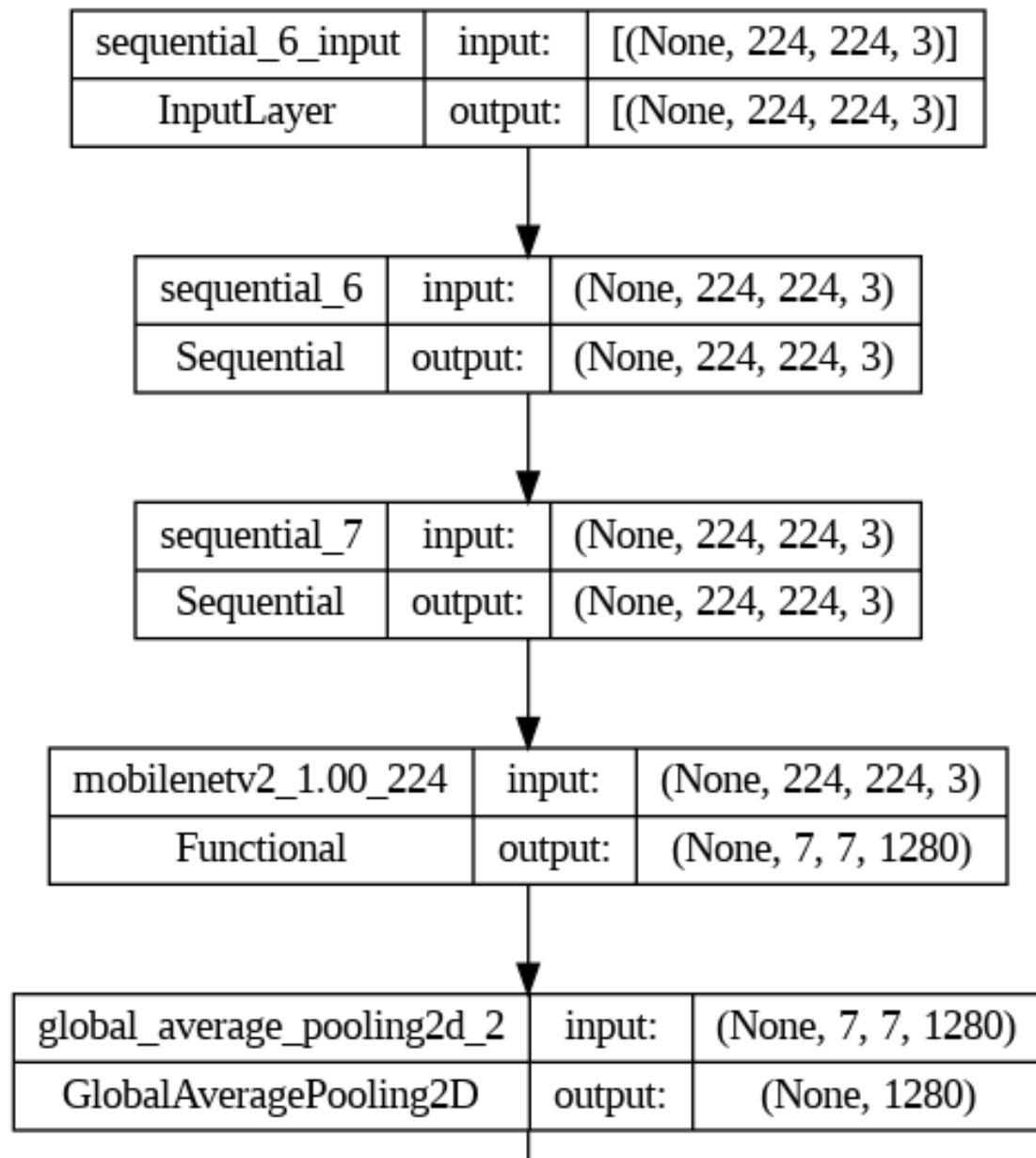


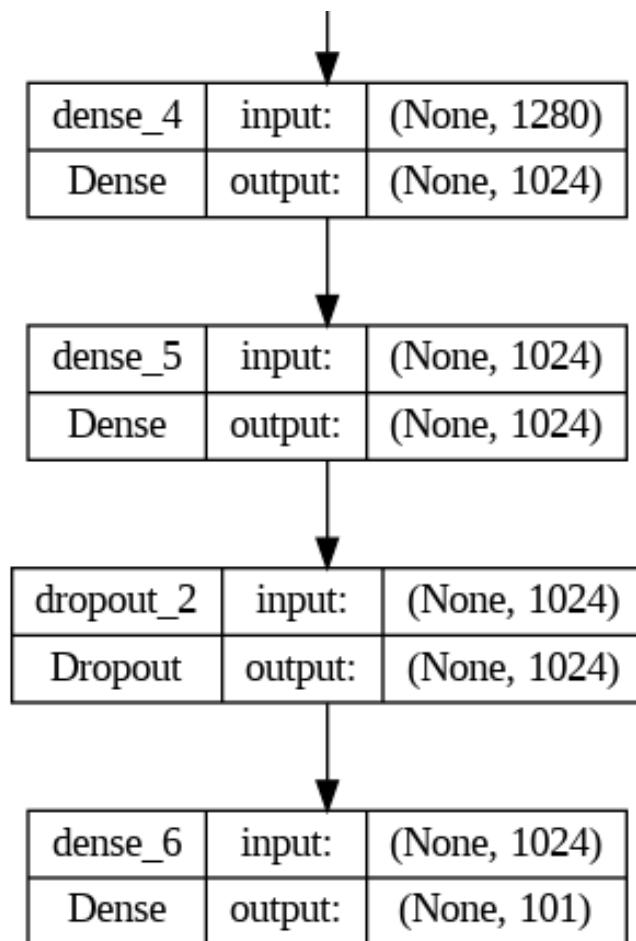
#### 4.4.2. Tuned base model

Due to the bad performance in terms of accuracy and overfitting we decided to make a hyper-parameters tuning to find the better ones and see the results that we can obtain. The hyperparameters that we have tried to tune are the following:

- Number of Dense hidden layers
- Dense hidden layer number of neurons
- Dropout percentage
- Learning rate

At the end of the tuning we have build the following network:



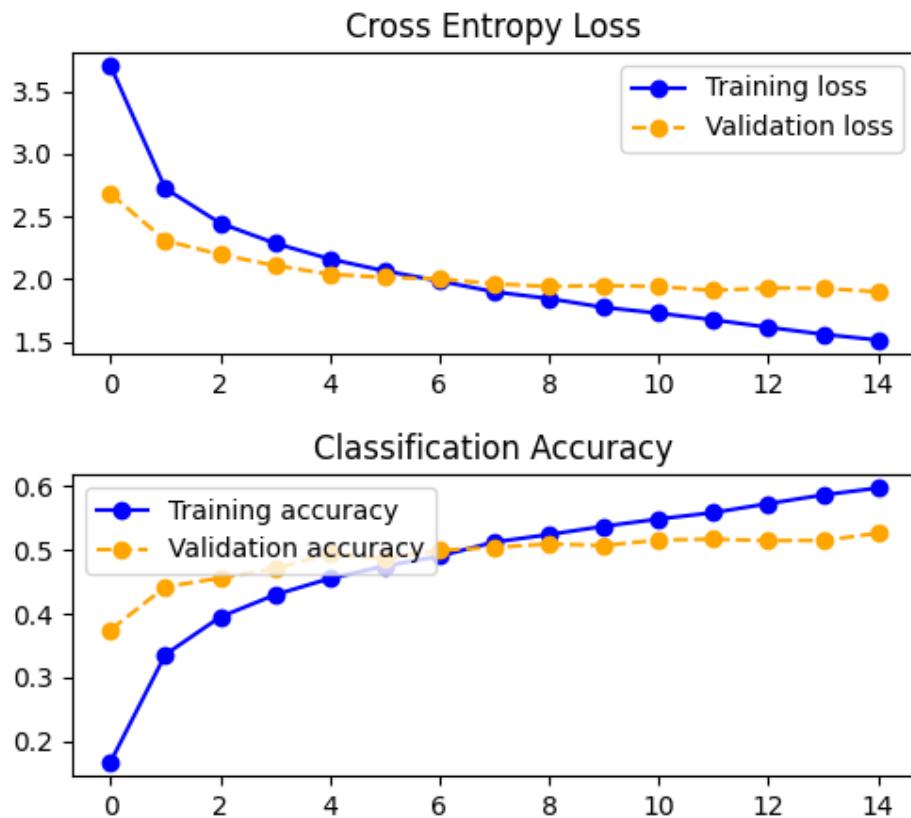


**Total params:** 4,722,853  
**Trainable params:** 2,464,869  
**Non-trainable params:** 2,257,984

Since we use an early stop callback to stop the training when there is no improvement in the accuracy on the validation set with a patience of 3, we need to retrain the network with the best number of epochs. The values reported in the following are the ones that we get with the network trained with the best number of epochs. The same procedure is applied for all the experiments of this section.

#### 4.4.3. Experiment 1 - Tuned freezed model

The first experiment is done by using the optimal hyper-parameters found in the previous experiment.



As we can see, this network has performance far better than the previous one: loss and accuracy have a much more consistent trend.

Best epoch	Validation accuracy	Test accuracy	Validation loss	Test loss
15	0.5258	0.5628	1.8987	1.6687



#### 4.4.4. Fine tuning

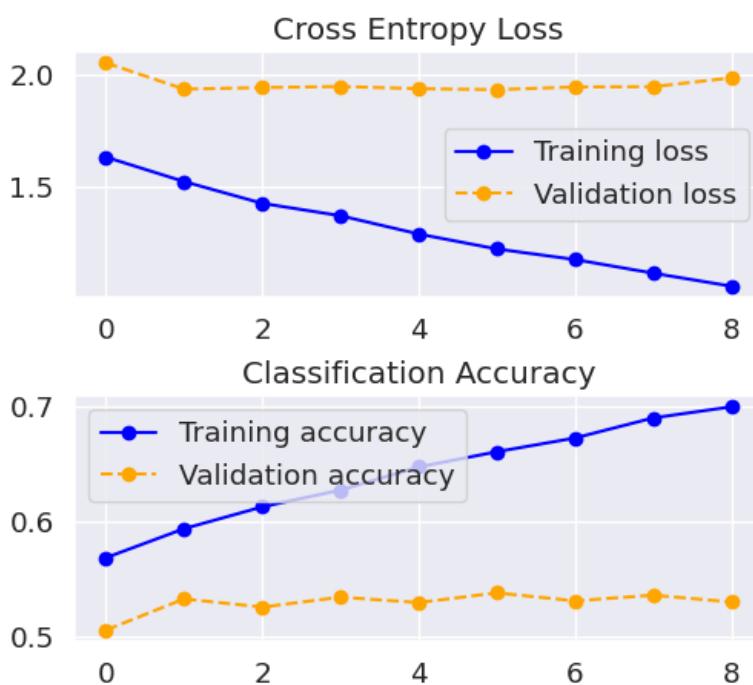
Fine tuning consists in unfreezing a few of the top layers of a frozen model base used for feature extraction, and jointly training both the classifier we added and these top layers. This is called "fine-tuning" because it slightly adjusts the more abstract representations of the model being reused, in order to make them more relevant for the problem at hand. Fine tuning can be useful when the new dataset is large and very different from the original

dataset like in this case. We can either train a CNN from scratch or fine-tuning a larger number of layers. Indeed, when we have a lot of data, we do not run the risk of overfitting. And when the data is different, we could benefit from the weight initialization of a pre-trained network and fine-tune even the earlier layers.

#### 4.4.4. Experiment 2 - Unfreeze 1

The third experiment is done by unfreezing the last convolutional block of MobileNetV2. The architecture is the same as previously, except that the total number of parameters is 4,722,853 (2,877,029 trainable and 1,845,824 non-trainable).

Layer (type)	Output Shape	Param #
<hr/>		
sequential_6 (Sequential)	(None, 224, 224, None)	0
sequential_7 (Sequential)	(None, 224, 224, None)	0
mobilenetv2_1.00_224 (Functional)	(None, 7, 7, 1280)	2257984
global_average_pooling2d_2 (GlobalAveragePooling2D)	(None, 1280)	0
dense_4 (Dense)	(None, 1024)	1311744
dense_5 (Dense)	(None, 1024)	1049600
dropout_2 (Dropout)	(None, 1024)	0
dense_6 (Dense)	(None, 101)	103525
<hr/>		
Total params: 4,722,853		
Trainable params: 2,877,029		
Non-trainable params: 1,845,824		



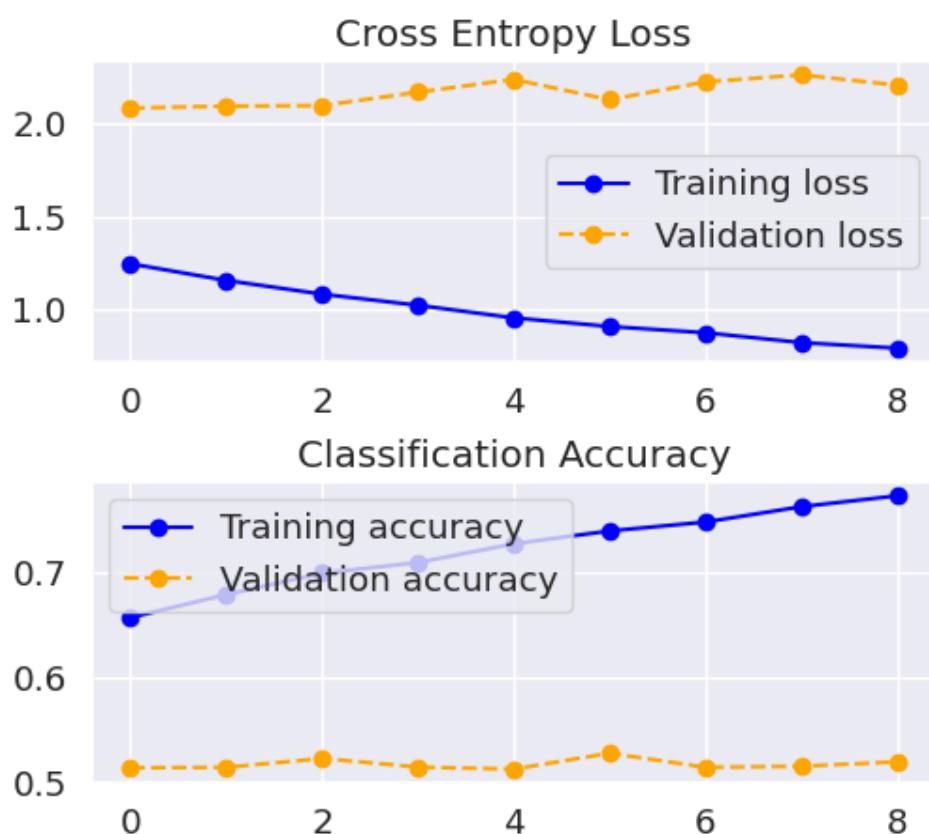
Best epoch	Validation accuracy	Test accuracy	Validation loss	Test loss
9	0.5298	0.5864	1.9836	1.6442



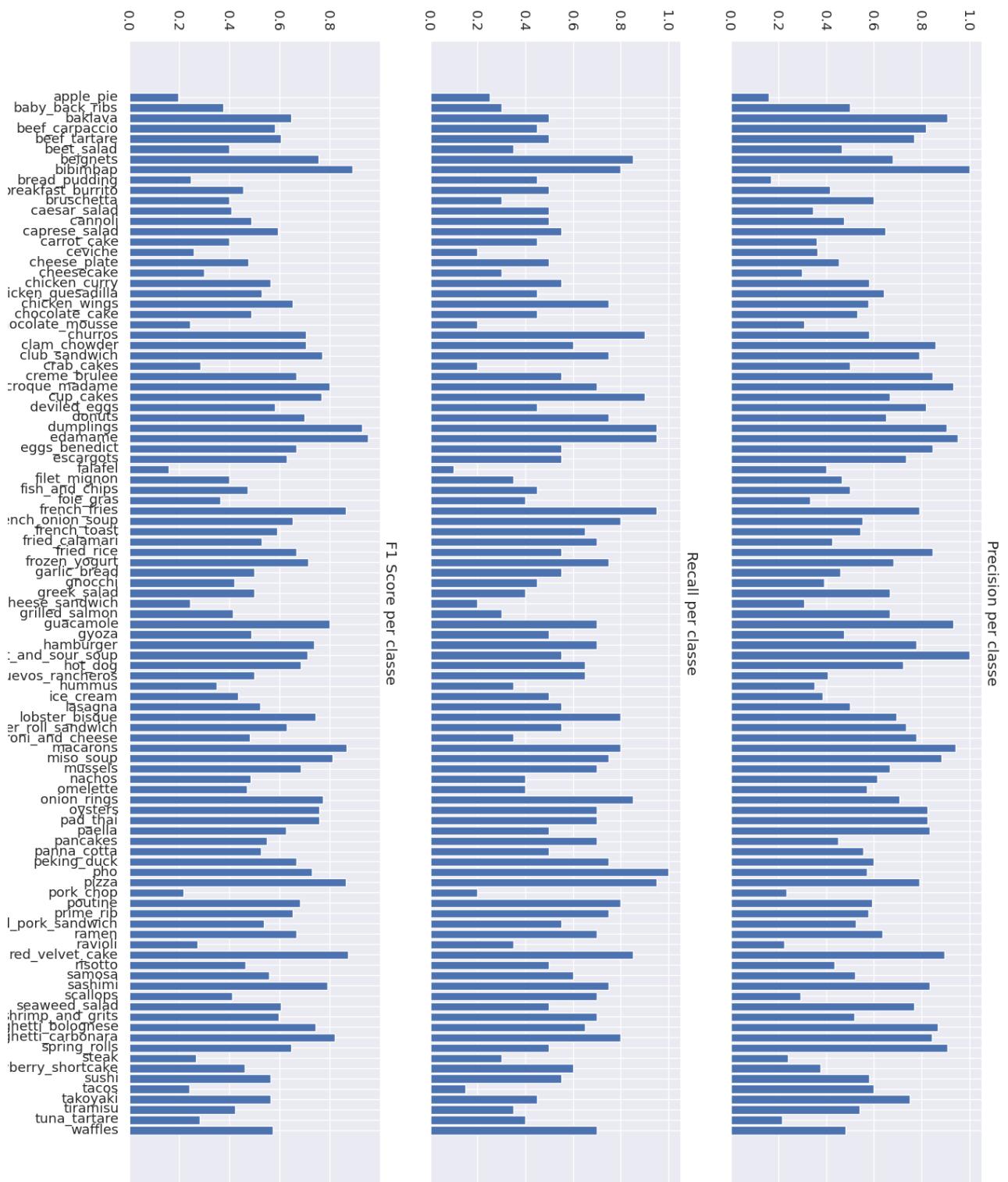
#### 4.4.5. Experiment 3 - Unfreeze 2

The fourth experiment is done by unfreezing the 2 last convolutional blocks of MobileNetV2. The architecture is the same as previously, except that the total number of parameters is 4,722,853 (3,184,869 trainable and 1,537,984 non trainable).

Layer (type)	Output Shape	Param #
<hr/>		
sequential_6 (Sequential)	(None, 224, 224, None)	0
sequential_7 (Sequential)	(None, 224, 224, None)	0
mobilenetv2_1.00_224 (Functional)	(None, 7, 7, 1280)	2257984
global_average_pooling2d_2 (GlobalAveragePooling2D)	(None, 1280)	0
dense_4 (Dense)	(None, 1024)	1311744
dense_5 (Dense)	(None, 1024)	1049600
dropout_2 (Dropout)	(None, 1024)	0
dense_6 (Dense)	(None, 101)	103525
<hr/>		
Total params: 4,722,853		
Trainable params: 3,184,869		
Non-trainable params: 1,537,984		



Best epoch	Validation accuracy	Test accuracy	Validation loss	Test loss
9	0.5198	0.5643	2.2108	1.8613



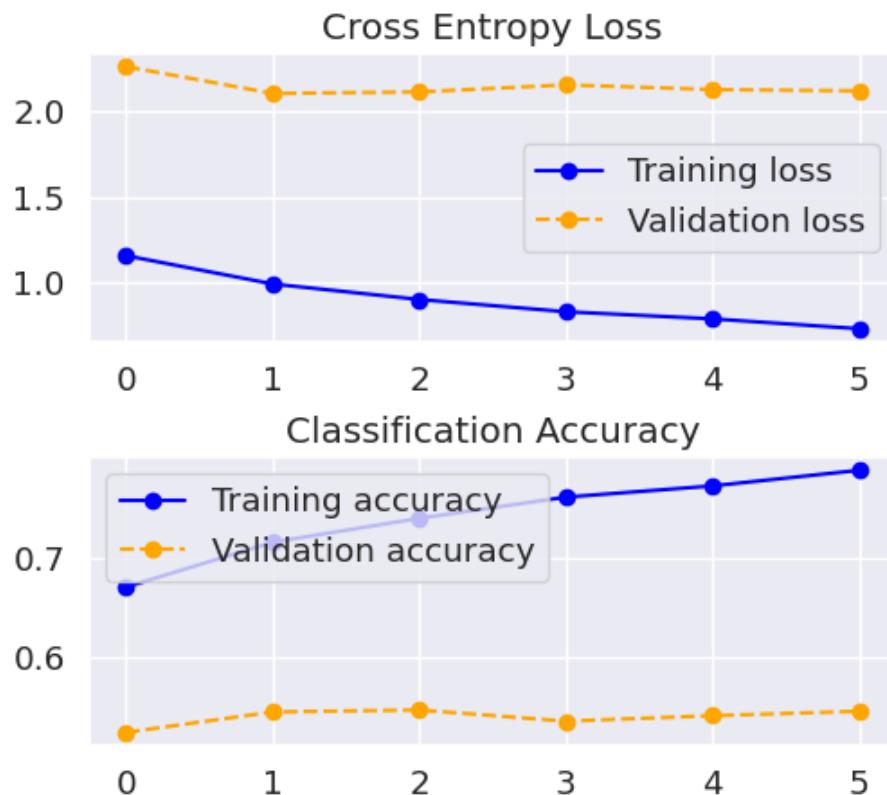
#### 4.4.6. Experiment 4 - Unfreeze 3

The fifth experiment is done by unfreezing the 3 last convolutional blocks of MobileNetV2. The architecture is the same as previously, except that the total number of parameters is 2,965,669 (1,593,765 trainable and 1,371,904 non-trainable).

Layer (type)	Output Shape	Param #
sequential_6 (Sequential)	(None, 224, 224, None)	0
sequential_7 (Sequential)	(None, 224, 224, None)	0
mobilenetv2_1.00_224 (Functional)	(None, 7, 7, 1280)	2257984
global_average_pooling2d_2 (GlobalAveragePooling2D)	(None, 1280)	0
dense_4 (Dense)	(None, 1024)	1311744
dense_5 (Dense)	(None, 1024)	1049600
dropout_2 (Dropout)	(None, 1024)	0
dense_6 (Dense)	(None, 101)	103525

Total params:	4,722,853
Trainable params:	3,350,949
Non-trainable params:	1,371,904

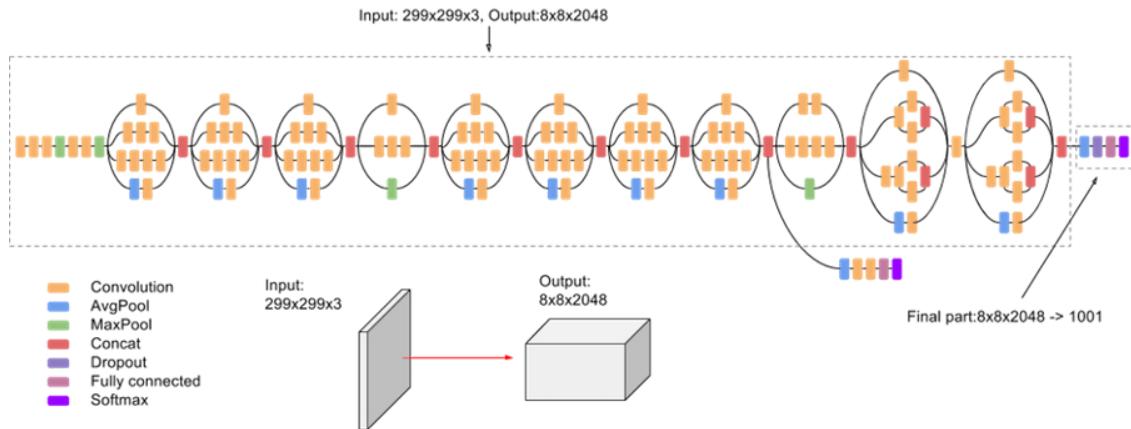


Best epoch	Validation accuracy	Test accuracy	Validation loss	Test loss
6	0.5469	0.5831	2.1238	1.8382



## 4.5. InceptionV3

InceptionV3 is a convolutional neural network for assisting in image analysis and object detection and got its start as a module for GoogLeNet. It is the third edition of Google's Inception Convolutional Neural Network, originally introduced during the ImageNet Recognition Challenge.



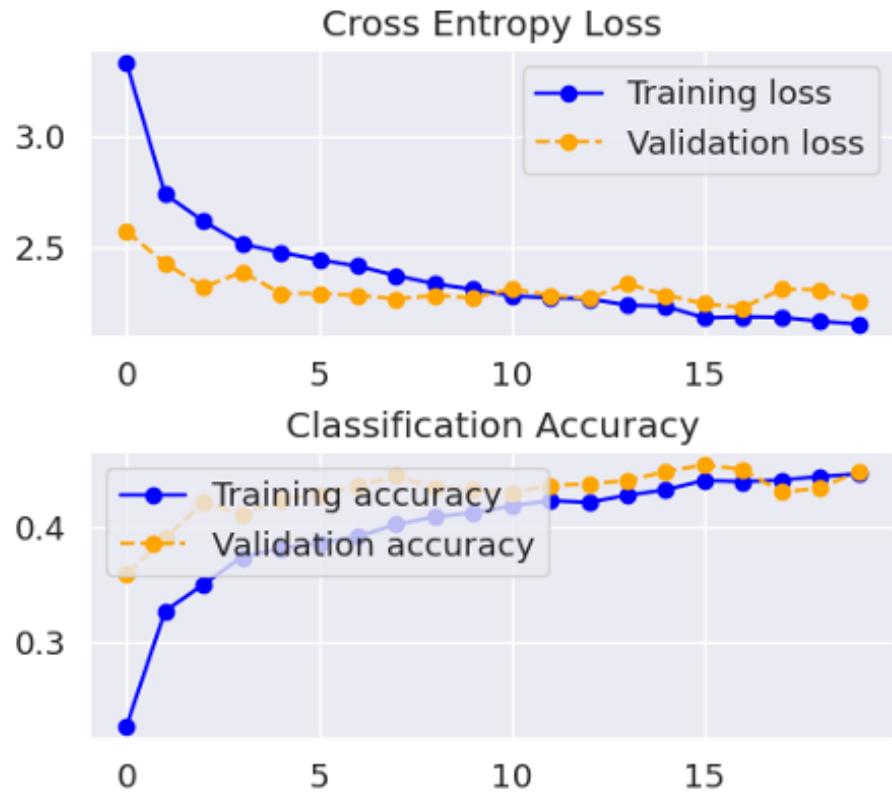
The third version was introduced by Szegedy et al. in Rethinking the Inception Architecture for Computer Vision. Makes several improvements from the previous models including using Label Smoothing, factorized 7 x 7 convolutions, and the use of an auxiliary classifier to propagate label information lower down the network (along with the use of batch normalization for layers in the sidehead).

### 4.5.1 Experiment 0 - Base model

The experiment was performed by using as base model the InceptionV3 (with frozen weights) and placing on top of this base the following layers: GlobalAveragePooling2D, Dense, Dropout and another Dense as an output layer.

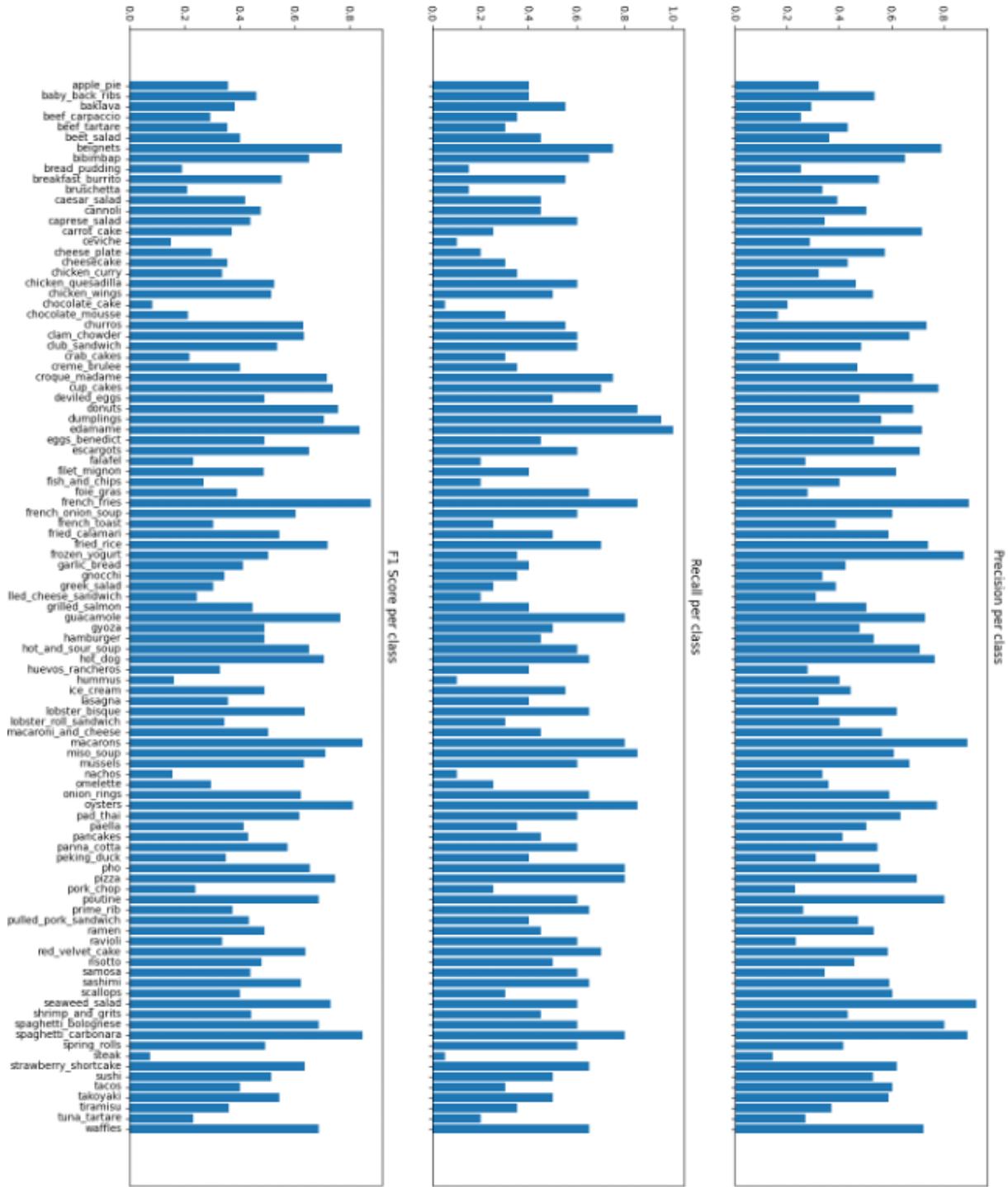
Layer (type)	Output Shape	Param #
<hr/>		
sequential (Sequential)	(None, 224, 224, 3)	0
sequential_1 (Sequential)	(None, 224, 224, 3)	0
inception_v3 (Functional)	(None, 5, 5, 2048)	21802784
global_average_pooling2d (GlobalAveragePooling2D)	(None, 2048)	0
dense (Dense)	(None, 1024)	2098176
dropout (Dropout)	(None, 1024)	0
dense_1 (Dense)	(None, 101)	103525
<hr/>		
Total params: 24,004,485		
Trainable params: 2,201,701		
Non-trainable params: 21,802,784		

The above pictures show the different layers of the model and the trainable (and non-trainable) params.



After 16 epochs, we observe overfitting: the accuracy on the training set is larger than the one on the validation set. And the accuracy on the validation set does not increase significantly anymore, which means that the network starts to memorize the training data.

Best epoch	Validation accuracy	Test accuracy	Validation loss	Test loss
20	0.44	0.48	2.26	2.02



### 4.5.2. Experiment 1 - Unfreeze 1

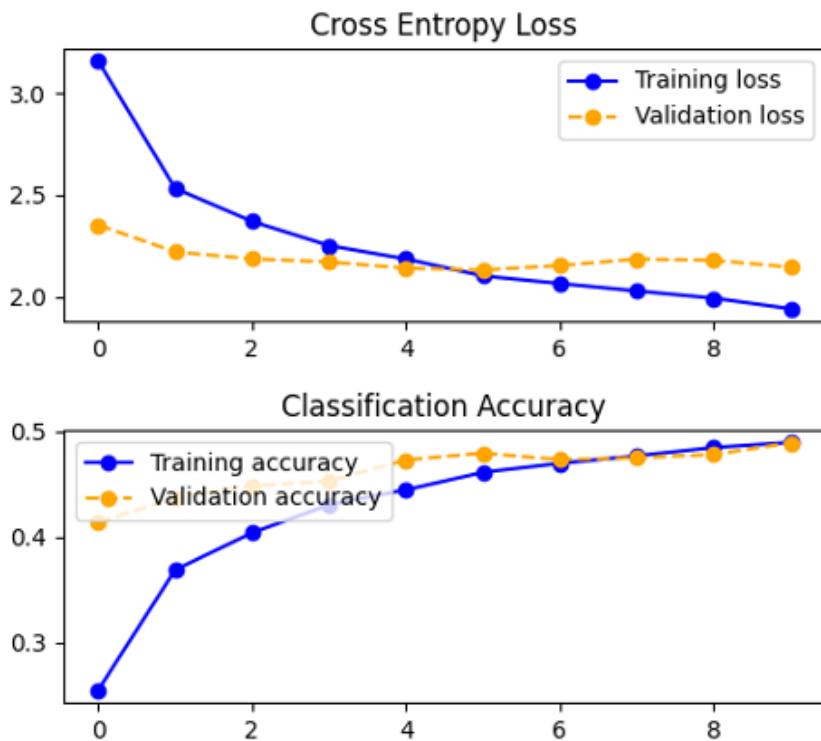
From the following experiment to the last one (Experiment 3) was performed the fine-tuning. The process of fine-tuning consists of unfreezing some layers of the model (in particular the one of the InceptionV3 that was chosen as our base).

Layer (type)	Output Shape	Param #
sequential (Sequential)	(None, 224, 224, 3)	0
sequential_1 (Sequential)	(None, 224, 224, 3)	0
inception_v3 (Functional)	(None, 5, 5, 2048)	21802784
global_average_pooling2d (GlobalAveragePooling2D)	(None, 2048)	0
dense (Dense)	(None, 1024)	2098176
dropout (Dropout)	(None, 1024)	0
dense_1 (Dense)	(None, 101)	103525

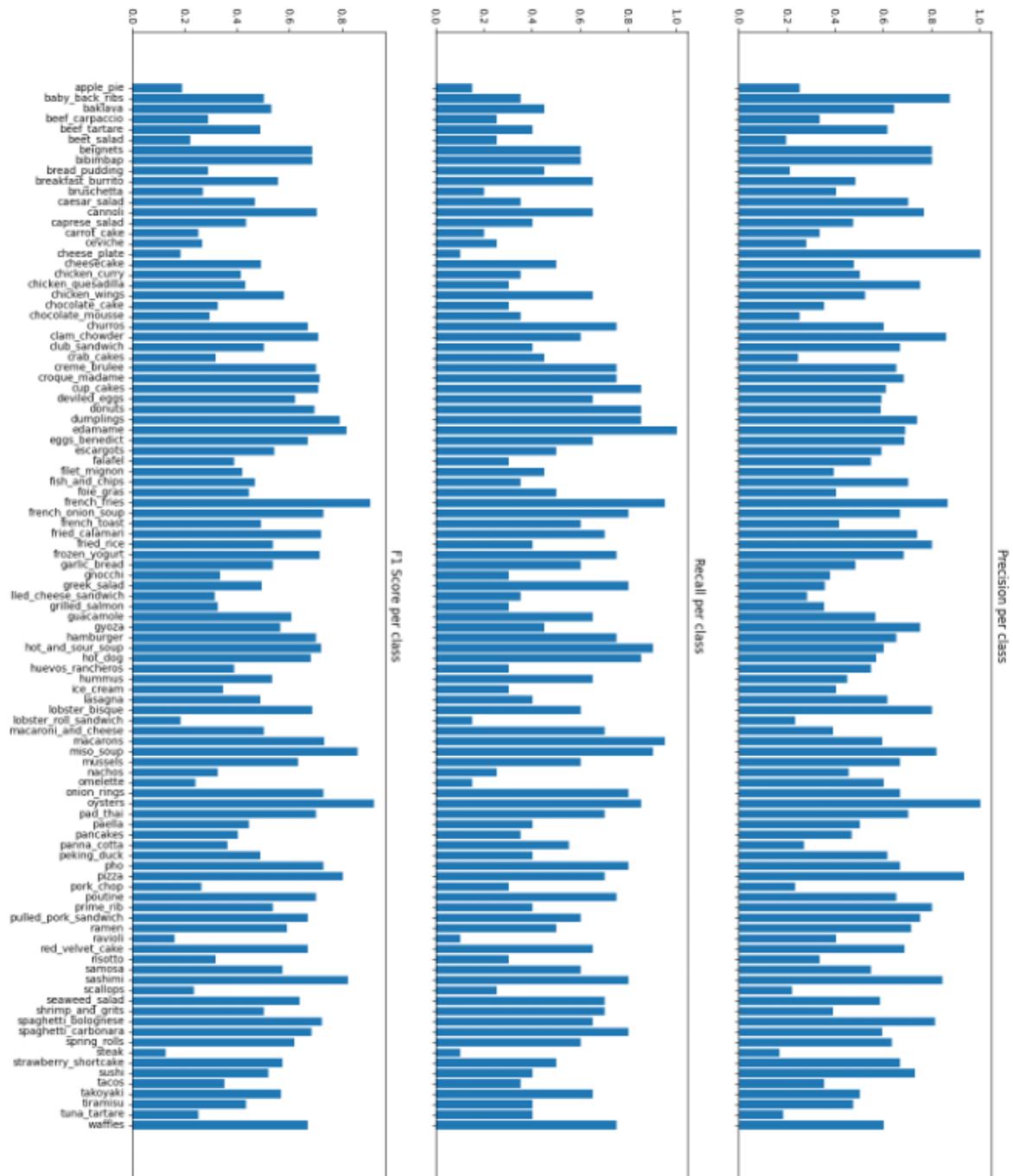
Total params:	24,004,485
Trainable params:	2,595,429
Non-trainable params:	21,409,056

The above pictures show the different layers of the model and the trainable (and non-trainable) params.



After 8 epochs, we observe overfitting: the accuracy on the training set is larger than the one on the validation set. And the accuracy on the validation set does not increase significantly anymore, which means that the network starts to memorize the training data.

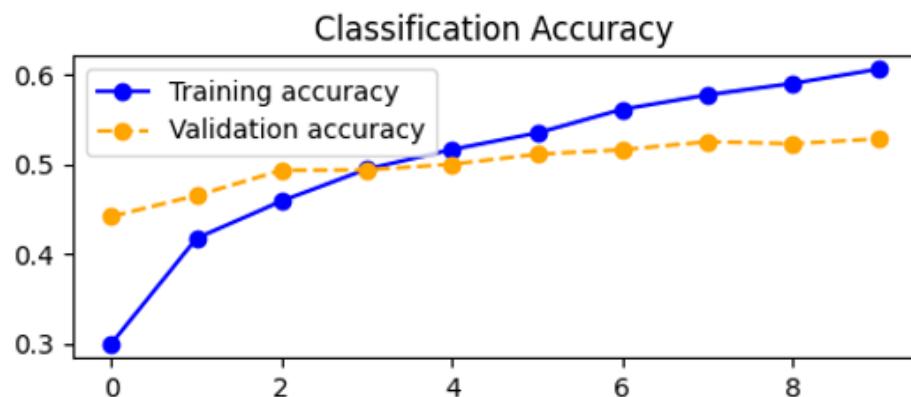
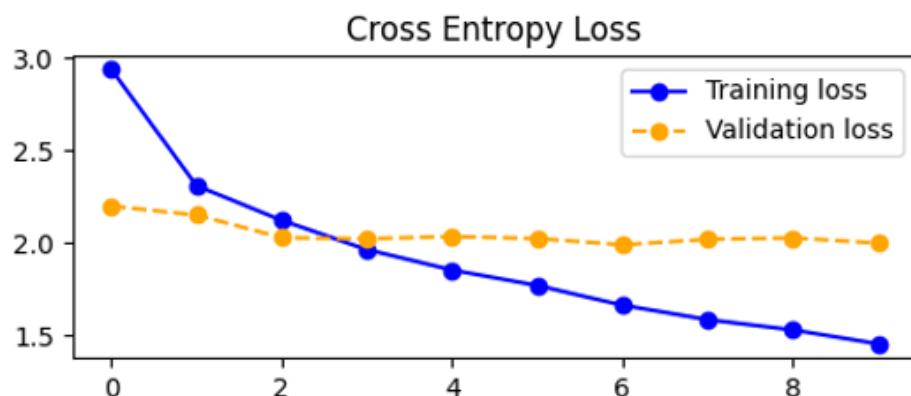
Best epoch	Validation accuracy	Test accuracy	Validation loss	Test loss
10	0.48	0.52	2.14	1.80



### 4.5.3. Experiment 2 - Unfreeze 2

Following with experiment 2 was performed the unfreezing of other layers as we can see in the picture below.

Layer (type)	Output Shape	Param #
sequential (Sequential)	(None, 224, 224, 3)	0
sequential_1 (Sequential)	(None, 224, 224, 3)	0
inception_v3 (Functional)	(None, 5, 5, 2048)	21802784
global_average_pooling2d (GlobalAveragePooling2D)	(None, 2048)	0
dense (Dense)	(None, 1024)	2098176
dropout (Dropout)	(None, 1024)	0
dense_1 (Dense)	(None, 101)	103525
<hr/>		
Total params: 24,004,485		
Trainable params: 3,252,325		
Non-trainable params: 20,752,160		



After 4 epochs, we observe overfitting: the accuracy on the training set is larger than the one on the validation set. And the accuracy on the validation set does not increase significantly anymore, which means that the network starts to memorize the training data.

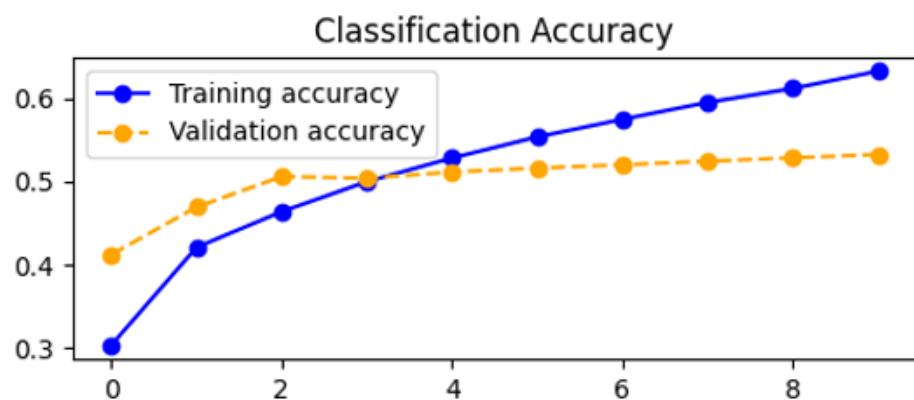
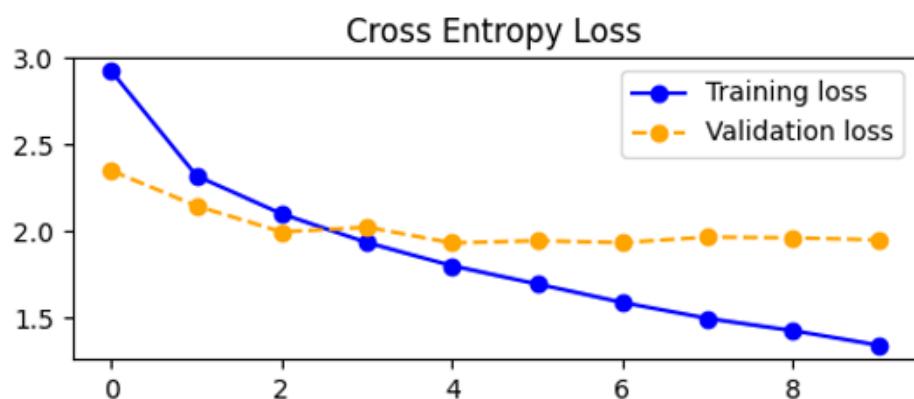
Best epoch	Validation accuracy	Test accuracy	Validation loss	Test loss
10	0.60	0.57	1.98	1.65



#### 4.5.4. Experiment 3 - Unfreeze 3

In the last experiment covering the fine-tuning we can do some math and see that we moved from experiment 1 with a total of 2,595,429 trainable params to the model of this experiment that has a total of 5,021,797 trainable params.

Layer (type)	Output Shape	Param #
sequential (Sequential)	(None, 224, 224, 3)	0
sequential_1 (Sequential)	(None, 224, 224, 3)	0
inception_v3 (Functional)	(None, 5, 5, 2048)	21802784
global_average_pooling2d (GlobalAveragePooling2D)	(None, 2048)	0
dense (Dense)	(None, 1024)	2098176
dropout (Dropout)	(None, 1024)	0
dense_1 (Dense)	(None, 101)	103525
<hr/>		
Total params: 24,004,485		
Trainable params: 5,021,797		
Non-trainable params: 18,982,688		



After 5 epochs, we observe overfitting: the accuracy on the training set is larger than the one on the validation set. And the accuracy on the validation set does not increase significantly anymore, which means that the network starts to memorize the training data.

Best epoch	Validation accuracy	Test accuracy	Validation loss	Test loss
10	0.53	0.58	1.94	1.64

