Subject: Advanced Programming

Instructor: Sultanmurat Yeleu

Group: IT-2101

Members: Azim Kassymbayev, Arman Karimov

Food recognition model

Table of contents

Introduction

Problem

Most of the people can't control what they are eating or how much nutrition their food has or they want to know the name of the food they've seen. In those situations we prepared the food recognition model. We want to make a model that will recognize types of food based on image. It is a test model that only recognises type of food, but with further development, the application that provides info about nutritional qualities of food can be implemented in fitness and diet. So people would be able to control their diet. Lots of people nowadays have issues with weight control, so more people try to control their weight with diets. For them, an application that identifies food by photo and provides useful info about it is a handful. For example, if an image of the cake named "tiramisu" will be inputed there are per 100g: 283 calories, 18,2g fat, 24,41g carbs, 4,77g protein.

Literature review (another solutions)

https://github.com/ivanDonadello/Food-Categories-Classification

https://tfhub.dev/google/aiy/vision/classifier/food_V1/1

https://github.com/MaharshSuryawala/Food-Image-Recognition

Here you can see some useful links and examples of models that we want to achieve. But we would like to add some more opportunities for those models, like were written earlier nutrition indicators.

Current work

Github link: https://github.com/Stalkover/food-10

Youtube link: https://youtu.be/BrRFQju_-ck

Telegram bot link: @food101bot

Data and Methods

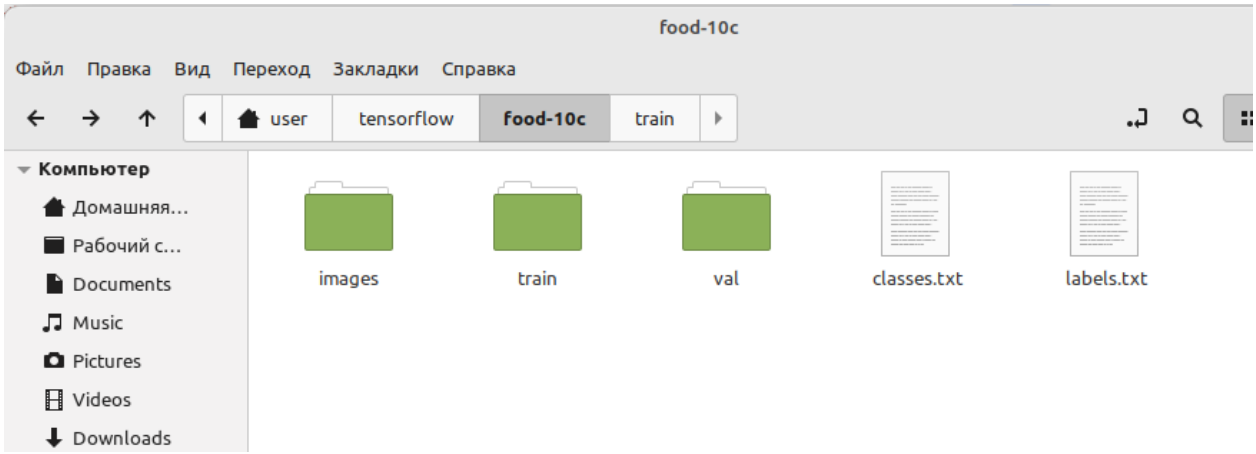
Dataset description

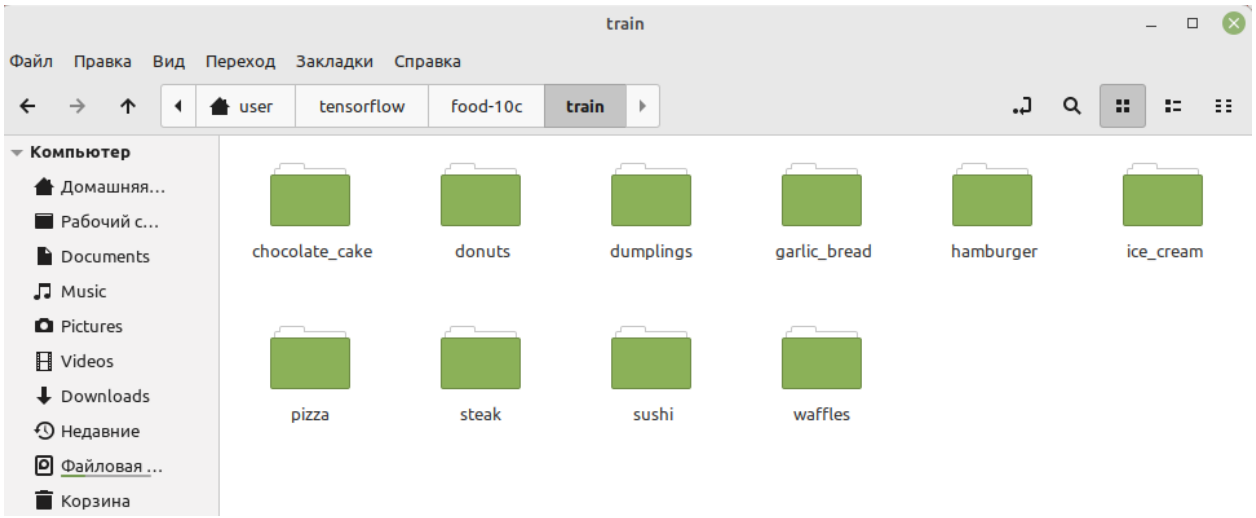
Initially we planned to train model on food-101 dataset with 101000 images, but it took a lot of time to train. We decided to take 10 labels and make a light version of the dataset to train and experiment with.

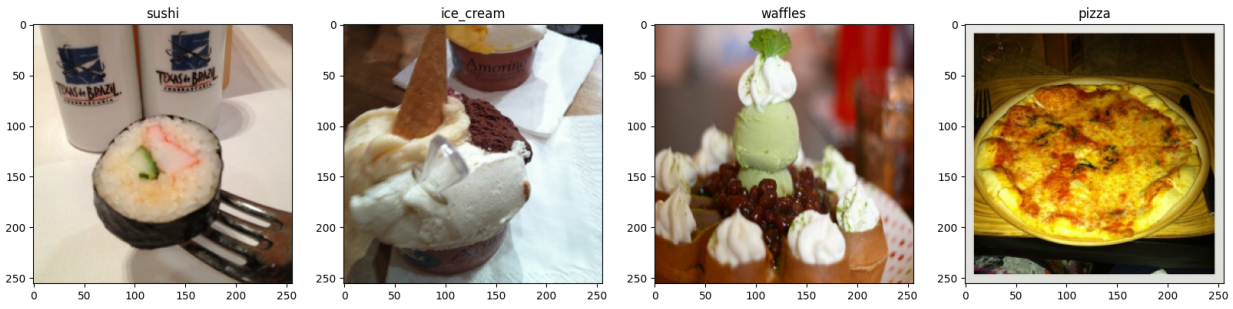
Database contains 10 labels with 1000 photos for each. We used a partition: 8000 for training, 2000 for validation.

We used Jupiter notebook to run Google Colab locally, and installed food-101 dataset.

Then we made food-10 dataset, choosing 10 labels and partitioning them so that the train folder has 80% of photos, and val has the rest 20%.





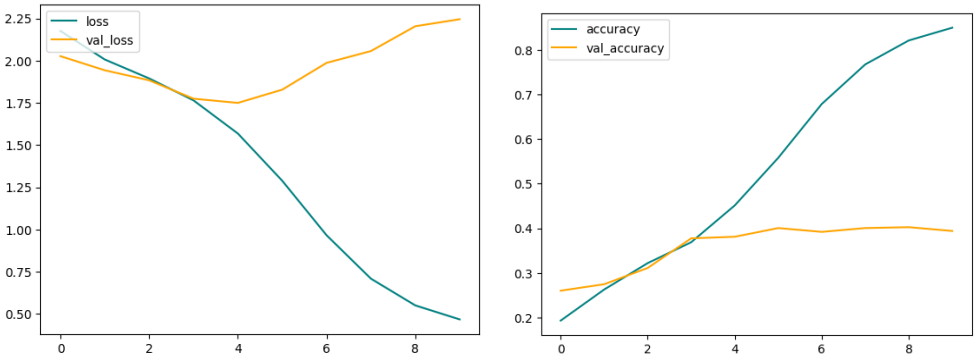


examples of images

Testing

Simple CNN model

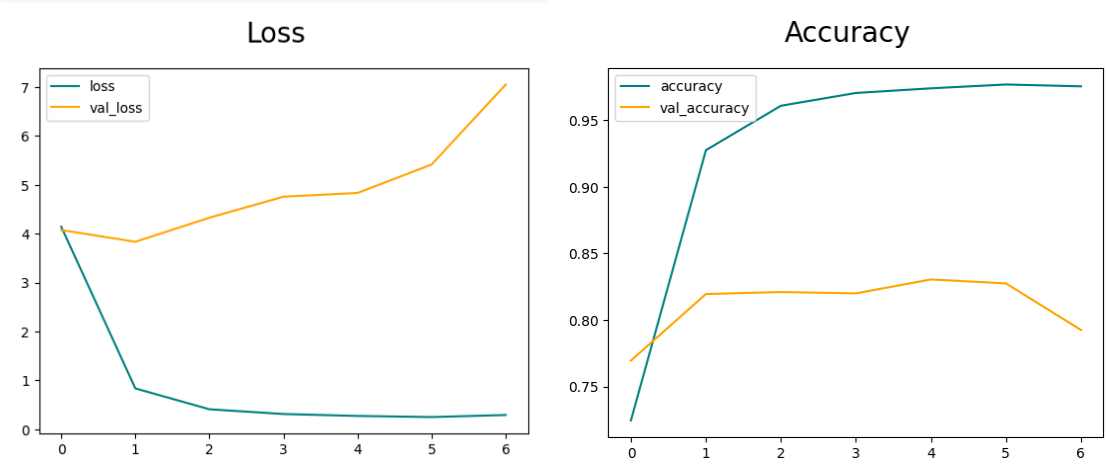
```
model = tf.keras.Sequential()

model.add(Conv2D(16, (3, 3), 1, activation='relu', input_shape=(256, 256, 3)))
model.add(MaxPooling2D())

model.add(Conv2D(32, (3, 3), 1, activation='relu'))
model.add(MaxPooling2D())

model.add(Conv2D(16, (3, 3), 1, activation='relu'))
model.add(MaxPooling2D())

model.add(Flatten())

model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(
    train,
    epochs=10,
    callbacks=[early_stopping],
    validation_data=val

)
```



Results: overfitting from 3 epoch

MobileNetV2 model (transfer learning)

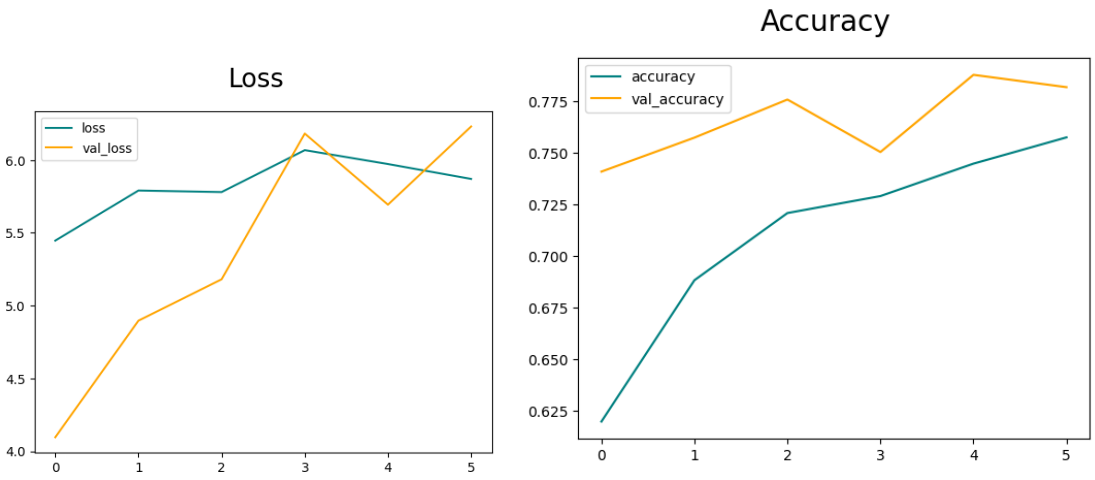
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 mobilenetv2_1.00_224 (Funct  (None, 8, 8, 1280)       2257984
 ional)

 flatten (Flatten)           (None, 81920)             0

 dense (Dense)               (None, 10)                819210

=================================================================
Total params: 3,077,194
Trainable params: 819,210
Non-trainable params: 2,257,984
```



Results: fast overfitting

MobileNet with data augmentation

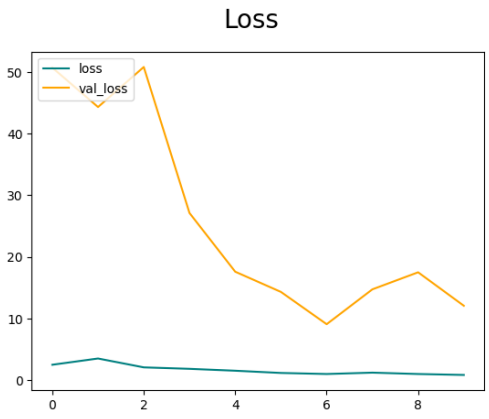
```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    vertical_flip=True,
    fill_mode='nearest'
)

val_datagen = ImageDataGenerator(rescale=1./255)
```

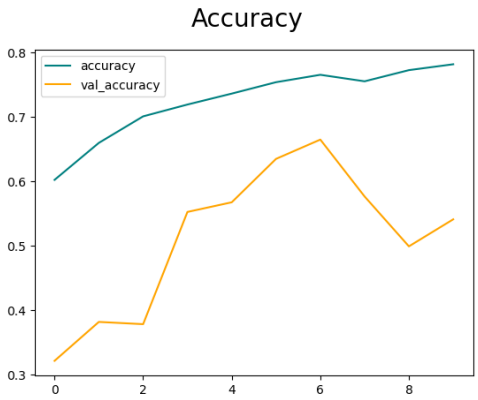


Results: overfitting, using data augmentation increased loss and decreased accuracy metrics, but didn't affect validation values

MobileNet, data augmentation, fine-tuning

```
base_model = tf.keras.applications.MobileNetV2(input_shape=(224, 224, 3), include_top=False, weights='imagenet')

for layer in base_model.layers[:-5]:
    layer.trainable = False
```

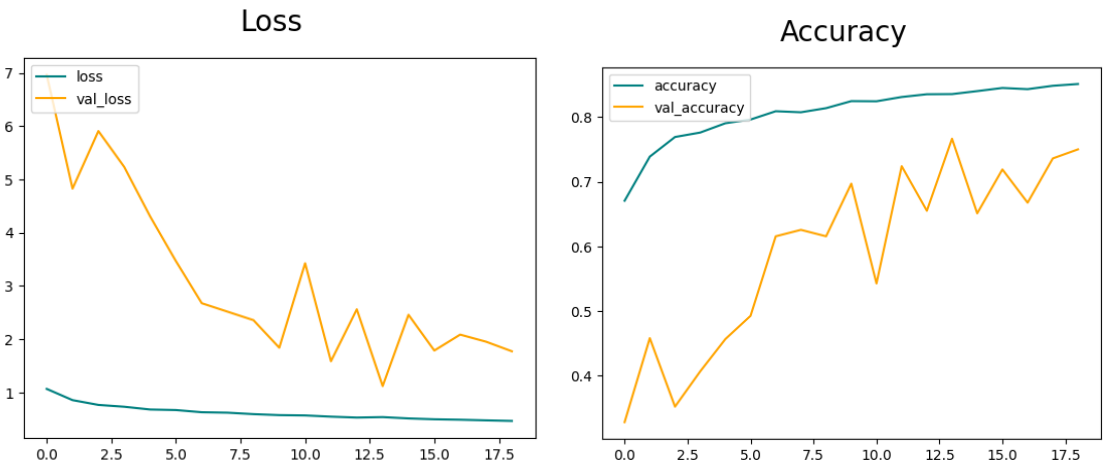




Results: high validation loss, slow accuracy growth, no overfitting

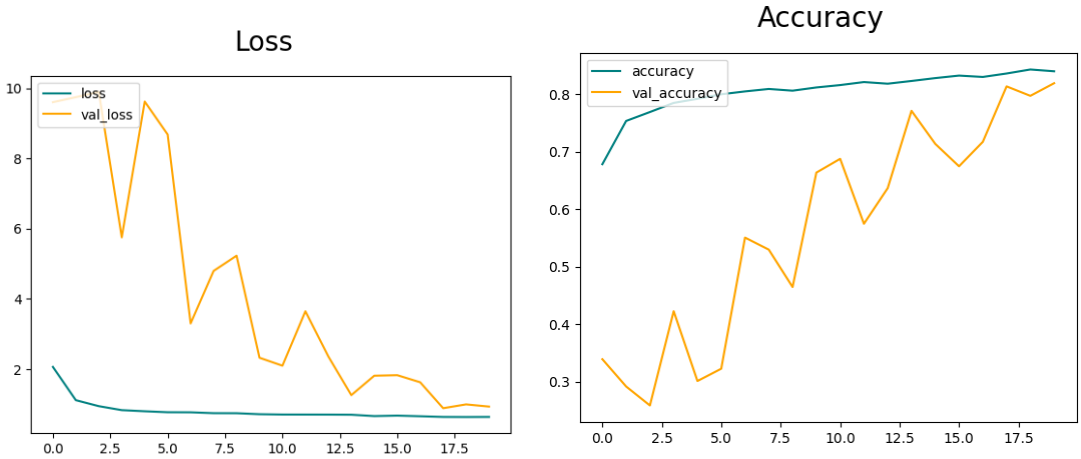
Previous setup with dropout layer

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 mobilenetv2_1.00_224 (Funct  (None, 7, 7, 1280)       2257984
 ional)

 global_average_pooling2d (G  (None, 1280)             0
 lobalAveragePooling2D)

 dense (Dense)               (None, 128)               163968

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 2,423,242
Trainable params: 885,258
Non-trainable params: 1,537,984
```



Results: more smooth accuracy increase

Previous setup with l2 regularizers



Results: fast training progress

2.4) Analysis

Transfer learning gave quick results compared to custom model, fine-tuning allowed to make model suitable for current task, which made training faster. We were able to avoid data augmentation by using different techniques, including: dropout, data augmentation, l2 regularizers, early stopping.

By the results option with l2 regularizers seems most accurate, it was able to achieve high results in a short amount of time, so we will use it as the main model.

Description of the ML model

```
model = tf.keras.Sequential()
model.add(base_model),
model.add(tf.keras.layers.GlobalAveragePooling2D()),
model.add(tf.keras.layers.Dense(128, activation='relu', kernel_regularizer=regularizers.l2(0.01))),
model.add(Dropout(0.5)),
model.add(tf.keras.layers.Dense(10, activation='softmax'))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 mobilenetv2_1.00_224 (Funct  (None, 7, 7, 1280)       2257984
 ional)

 global_average_pooling2d (G  (None, 1280)             0
 lobalAveragePooling2D)

 dense (Dense)               (None, 128)               163968

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 2,423,242
Trainable params: 885,258
Non-trainable params: 1,537,984
```

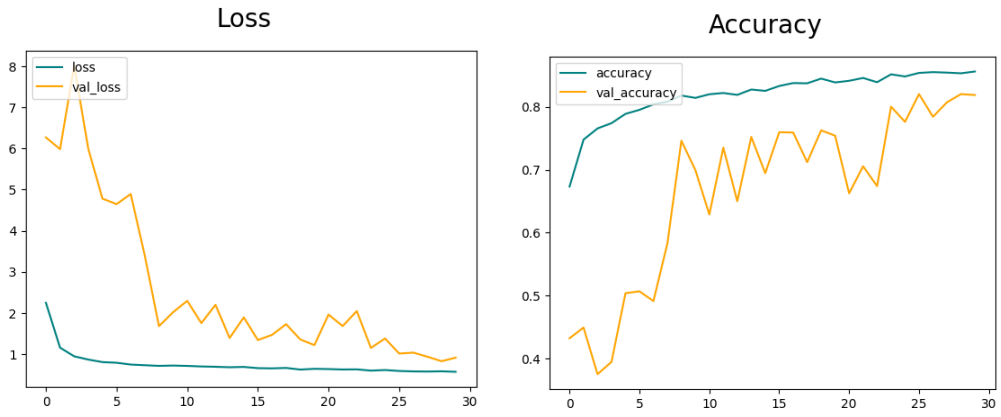
```
model.compile(optimizer=tf.keras.optimizers.Adam(lr=1e-5),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

The final model uses MobileNetV2 as a pre-trained CNN model suitable for image classification. It fine tunes the last 5 layers of the model, and compiles the model with 0.00001 learning rate to prevent overfitting.

Results

The model was trained in overall for 30 epochs

```
Epoch 13/30
250/250 [==============================] - 117s 466ms/step - loss: 0.6869 - accuracy: 0.8188 - val_loss: 2.1952 - val_accuracy: 0.6500
Epoch 14/30
250/250 [==============================] - 116s 463ms/step - loss: 0.6752 - accuracy: 0.8273 - val_loss: 1.3873 - val_accuracy: 0.7520
Epoch 15/30
250/250 [==============================] - 116s 465ms/step - loss: 0.6831 - accuracy: 0.8251 - val_loss: 1.8890 - val_accuracy: 0.6945
Epoch 16/30
250/250 [==============================] - 114s 457ms/step - loss: 0.6535 - accuracy: 0.8329 - val_loss: 1.3380 - val_accuracy: 0.7595
Epoch 17/30
250/250 [==============================] - 115s 460ms/step - loss: 0.6480 - accuracy: 0.8375 - val_loss: 1.4637 - val_accuracy: 0.7590
Epoch 18/30
250/250 [==============================] - 115s 460ms/step - loss: 0.6588 - accuracy: 0.8371 - val_loss: 1.7237 - val_accuracy: 0.7120
Epoch 19/30
250/250 [==============================] - 116s 466ms/step - loss: 0.6191 - accuracy: 0.8446 - val_loss: 1.3520 - val_accuracy: 0.7625
Epoch 20/30
250/250 [==============================] - 114s 455ms/step - loss: 0.6374 - accuracy: 0.8385 - val_loss: 1.2145 - val_accuracy: 0.7540
Epoch 21/30
250/250 [==============================] - 116s 464ms/step - loss: 0.6310 - accuracy: 0.8411 - val_loss: 1.9581 - val_accuracy: 0.6625
Epoch 22/30
250/250 [==============================] - 113s 453ms/step - loss: 0.6210 - accuracy: 0.8456 - val_loss: 1.6779 - val_accuracy: 0.7055
Epoch 23/30
250/250 [==============================] - 113s 451ms/step - loss: 0.6238 - accuracy: 0.8389 - val_loss: 2.0430 - val_accuracy: 0.6740
Epoch 24/30
250/250 [==============================] - 117s 467ms/step - loss: 0.5932 - accuracy: 0.8514 - val_loss: 1.1461 - val_accuracy: 0.8000
Epoch 25/30
250/250 [==============================] - 116s 463ms/step - loss: 0.6077 - accuracy: 0.8480 - val_loss: 1.3753 - val_accuracy: 0.7760
Epoch 26/30
250/250 [==============================] - 113s 450ms/step - loss: 0.5849 - accuracy: 0.8536 - val_loss: 1.0088 - val_accuracy: 0.8200
Epoch 27/30
250/250 [==============================] - 127s 508ms/step - loss: 0.5747 - accuracy: 0.8549 - val_loss: 1.0316 - val_accuracy: 0.7840
Epoch 28/30
250/250 [==============================] - 115s 460ms/step - loss: 0.5711 - accuracy: 0.8541 - val_loss: 0.9315 - val_accuracy: 0.8070
Epoch 29/30
250/250 [==============================] - 116s 462ms/step - loss: 0.5767 - accuracy: 0.8530 - val_loss: 0.8222 - val_accuracy: 0.8200
Epoch 30/30
250/250 [==============================] - 127s 509ms/step - loss: 0.5650 - accuracy: 0.8559 - val_loss: 0.9097 - val_accuracy: 0.8185
```



Final results

loss: 0.5650 - accuracy: 0.8559 - val_loss: 0.9097 - val_accuracy: 0.8185

At the end our accuracy test showed us a result of 81%, after many different tests, we achieved a good result.

Next steps

There are a lot we would like to add to our model, first of all adding more layers from original 101-food dataset, making custom food labels, implementing more complex techniques for data augmentation, also the accuracy could be higher over time and as was written earlier adding nutrition indicators for each label.