# Projekt semestralny IO - 21/22

## Wstęp

---

### O projekcie:

Tematem projektu jest rozpoznawanie jedzenia na zdjęciach poprzez modele uczenia maszynowego. Celem projektu jest wytrenowanie trzech różnych modeli klasyfikujących i wybór najkorzystniejszego.

---

### Tech stack:

Paczki:

- Tensorflow.keras
- Tensorflow-hub
- Matplotlib

Modele:

- ResNet_V2_50: link
- MobileNet_V3_100: link
- EfficientNet_V2_b0: link

Narzędzia:

- Google Colab: link, z włączoną optymalizacją GPU

---

### Baza danych

Wykorzystywana baza danych to **Food 101** (link) zawierająca 101 kategorii żywności, z łącznie 101 000 obrazami. Dla każdej klasy dostarczono 250 ręcznie zweryfikowanych obrazów testowych oraz 750 obrazów treningowych. Celowo, obrazy treningowe nie zostały oczyszczone, a zatem nadal zawierają pewną ilość szumu. Występują one głównie w postaci intensywnych kolorów i czasami błędnych etykiet. Wszystkie obrazy zostały przeskalowane tak, aby ich maksymalna długość boku wynosiła 512 pikseli.

# Opis eksperymentów

1. Pobranie bazy danych i wypakowanie jej do subfolderu `input`
2. Zapisanie stałych aplikacji zawierających: ścieżkę do zdjęć, oczekiwany format zdjęć oraz linki do modeli (z racji tego, że wykorzystujemy modele zewnętrzne).

```
# constants
data_dir = './input/images/'
IMAGE_SHAPE = (224, 224)
ResNet_V2_50 = 'https://tfhub.dev/google/imagenet/resnet_v2_50/classification/5'
MobileNet_V3_100 = "https://tfhub.dev/google/imagenet/mobilenet_v3_large_100_224/classification/5"
EfficientNet_V2_b0 = "https://tfhub.dev/google/imagenet/efficientnet_v2_imagenet1k_b0/classification/2"
```

3. Przygotowanie zbiorów treningowego i testowego z uwzgędnieniem ustalonego formatu zdjęć poprzez ImageDataGenerator pochodzący z keras.

```
dataGenerator = ImageDataGenerator(rescale=1./255, validation_split=0.2)
train_data = dataGenerator.flow_from_directory(
    data_dir,
    target_size=IMAGE_SHAPE,
    batch_size=32,
    class_mode='categorical',
    subset='training'
)

val_data = dataGenerator.flow_from_directory(
    data_dir,
    target_size=IMAGE_SHAPE,
    batch_size=32,
    class_mode='categorical',
    subset='validation'
)
```

4. Budowanie modeli, trening, ewaluacja
    1. Przygotowanie modeli wykorzystując model sekwencyjny pochodzący z keras ( Pozwala nam to na stworzenie modelu deep learning poprzez dodawanie do niego warstw. Każda jednostka w warstwie jest połączona z każdą jednostką w poprzedniej warstwie. )
    2. Kompilacja modeli z wykorzystaniem optymalizatora SGD (link), który charakteryzuje się stosunkowo szybki i dokładnym rozwiązaniem
    3. Trenowanie modeli na danych treningowych przez 10 epok
    4. Ewaulacja modeli danymi testowymi

```
model_MobileNet = tf.keras.Sequential([
    hub.KerasLayer(MobileNet_V3_100, input_shape=IMAGE_SHAPE+(3,), name="MobileNet_V3_100"),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(101, activation='softmax', name='Output_layer')
])
```

```
    hub.KerasLayer(EfficientNet_V2_b0, trainable=False, input_shape=IMAGE_SHAPE+(3,), name='EfficientNet_V2_b0'),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(101, activation='softmax', name='Output_layer')
])

model_EfficientNet.compile(
    optimizer=tf.keras.optimizers.SGD(),
    loss=tf.keras.losses.CategoricalCrossentropy(),
    metrics=['accuracy']
)

model_EfficientNet.summary()
efficientNet_model = model_EfficientNet.fit(train_data, epochs=10, verbose=1)
model_EfficientNet.evaluate(val_data)
```

```
model_EfficientNet = tf.keras.Sequential([
    hub.KerasLayer(EfficientNet_V2_b0, trainable=False, input_shape=IMAGE_SHAPE+(3,), name='EfficientNet_V2_b0'),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(101, activation='softmax', name='Output_layer')
])

model_EfficientNet.compile(
    optimizer=tf.keras.optimizers.SGD(),
    loss=tf.keras.losses.CategoricalCrossentropy(),
    metrics=['accuracy']
)

model_EfficientNet.summary()
efficientNet_model = model_EfficientNet.fit(train_data, epochs=10, verbose=1)
model_EfficientNet.evaluate(val_data)
```

```
model_ResNet = tf.keras.Sequential([
    hub.KerasLayer(ResNet_V2_50, trainable=False, input_shape=IMAGE_SHAPE+(3,), name='Resnet_V2_50'),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(101, activation='softmax', name='Output_layer')
])

model_ResNet.compile(
    optimizer=tf.keras.optimizers.SGD(),
    loss=tf.keras.losses.CategoricalCrossentropy(),
    metrics=['accuracy']
)


model_ResNet.summary()
resnet_model = model_ResNet.fit(train_data, epochs=10, verbose=1)
model_ResNet.evaluate(val_data)
```

# Wyniki

---

## MobileNet V3 100

```
_________________________________________________________________
Epoch 1/10
2525/2525 [==============================] - 535s 206ms/step - loss: 2.2135 - accuracy: 0.4615
Epoch 2/10
2525/2525 [==============================] - 454s 180ms/step - loss: 1.5480 - accuracy: 0.6002
Epoch 3/10
2525/2525 [==============================] - 449s 178ms/step - loss: 1.4110 - accuracy: 0.6337
Epoch 4/10
2525/2525 [==============================] - 452s 179ms/step - loss: 1.3320 - accuracy: 0.6519
Epoch 5/10
2525/2525 [==============================] - 460s 182ms/step - loss: 1.2769 - accuracy: 0.6657
Epoch 6/10
2525/2525 [==============================] - 459s 182ms/step - loss: 1.2354 - accuracy: 0.6755
Epoch 7/10
2525/2525 [==============================] - 461s 182ms/step - loss: 1.2015 - accuracy: 0.6850
Epoch 8/10
2525/2525 [==============================] - 463s 183ms/step - loss: 1.1734 - accuracy: 0.6929
Epoch 9/10
2525/2525 [==============================] - 461s 183ms/step - loss: 1.1496 - accuracy: 0.6965
Epoch 10/10
2525/2525 [==============================] - 459s 182ms/step - loss: 1.1280 - accuracy: 0.7019
632/632 [==============================] - 131s 206ms/step - loss: 1.4076 - accuracy: 0.6376
```

---

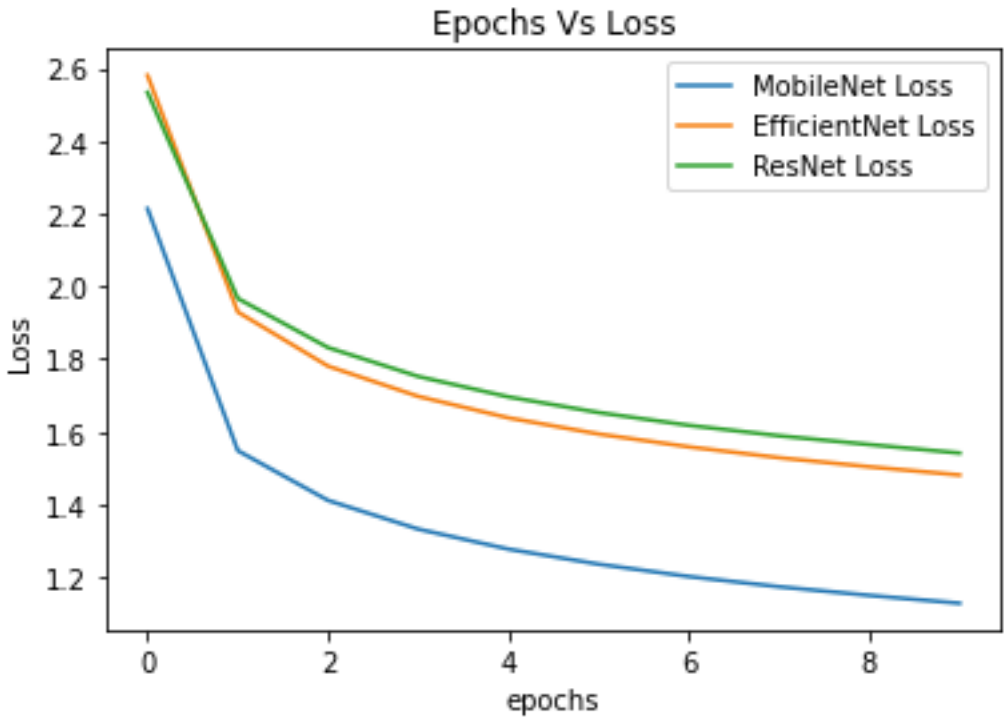
## EfficientNet V2 B0

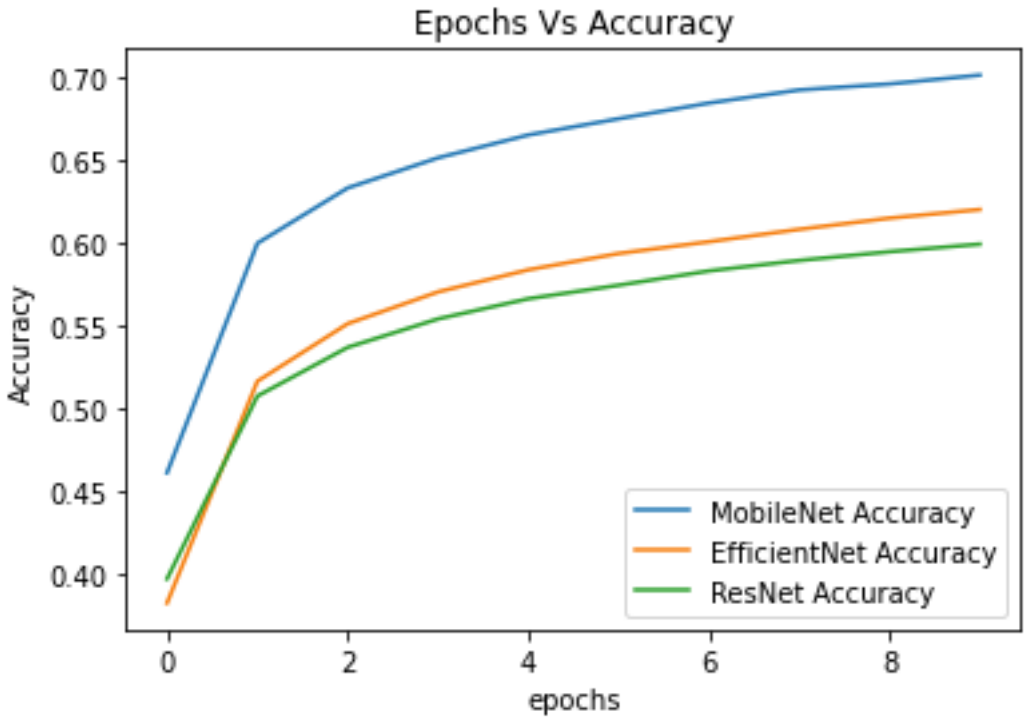
```
_________________________________________________________________
Epoch 1/10
2525/2525 [==============================] - 482s 188ms/step - loss: 2.5803 - accuracy: 0.3828
Epoch 2/10
2525/2525 [==============================] - 476s 189ms/step - loss: 1.9295 - accuracy: 0.5168
Epoch 3/10
2525/2525 [==============================] - 474s 188ms/step - loss: 1.7805 - accuracy: 0.5516
Epoch 4/10
2525/2525 [==============================] - 478s 189ms/step - loss: 1.6972 - accuracy: 0.5708
Epoch 5/10
2525/2525 [==============================] - 479s 190ms/step - loss: 1.6381 - accuracy: 0.5842
Epoch 6/10
2525/2525 [==============================] - 476s 188ms/step - loss: 1.5936 - accuracy: 0.5940
Epoch 7/10
2525/2525 [==============================] - 478s 189ms/step - loss: 1.5583 - accuracy: 0.6012
Epoch 8/10
2525/2525 [==============================] - 477s 189ms/step - loss: 1.5285 - accuracy: 0.6087
Epoch 9/10
2525/2525 [==============================] - 477s 189ms/step - loss: 1.5034 - accuracy: 0.6154
Epoch 10/10
2525/2525 [==============================] - 480s 190ms/step - loss: 1.4808 - accuracy: 0.6206
632/632 [==============================] - 121s 189ms/step - loss: 1.6689 - accuracy: 0.5766
```

## ResNet V2 50

```
_________________________________________________________________
Epoch 1/10
2525/2525 [==============================] - 544s 213ms/step - loss: 2.5337 - accuracy: 0.3973
Epoch 2/10
2525/2525 [==============================] - 542s 215ms/step - loss: 1.9667 - accuracy: 0.5078
Epoch 3/10
2525/2525 [==============================] - 543s 215ms/step - loss: 1.8315 - accuracy: 0.5373
Epoch 4/10
2525/2525 [==============================] - 542s 215ms/step - loss: 1.7520 - accuracy: 0.5545
Epoch 5/10
2525/2525 [==============================] - 542s 215ms/step - loss: 1.6958 - accuracy: 0.5667
Epoch 6/10
2525/2525 [==============================] - 543s 215ms/step - loss: 1.6527 - accuracy: 0.5748
Epoch 7/10
2525/2525 [==============================] - 540s 214ms/step - loss: 1.6175 - accuracy: 0.5835
Epoch 8/10
2525/2525 [==============================] - 539s 214ms/step - loss: 1.5887 - accuracy: 0.5898
Epoch 9/10
2525/2525 [==============================] - 535s 212ms/step - loss: 1.5646 - accuracy: 0.5951
Epoch 10/10
2525/2525 [==============================] - 535s 212ms/step - loss: 1.5412 - accuracy: 0.5997
632/632 [==============================] - 134s 210ms/step - loss: 1.8426 - accuracy: 0.5415
```

## Porównanie






---

## Interpretacja

Wszystkie trzy modele największy przyrost dokładności osiągnęły już w pierwszej epoce, a później kontynuowały wzrost parabolicznie. Każdy z modeli do ostatniej epoki osiągał wzrost dokładności i spadek start (chociaż przy małej zmianie), z czego można wnioskować, że istnieje jeszcze delikatny potencjał poprawy.  Najwyższą dokładność i najszybszy przyrost uczenia oraz najkrótsze epoki wykazał zoptymalizowany pod  pod urządzenia mobilne model **MobileNet V3 100.** W związku z powyższym warto go wskazać jako jako najlepszy wybór z testowanego zbioru.

# Bibliografia:


- https://github.com/stratospark/food-101-keras *
- https://towardsdatascience.com/training-machine-learning-models-online-for-free-gpu-tpu-enabled-5def6a5c1ce3 *


* dostęp: 25.01.2022r.