Overcoming NSP Noise

A project aimed to overcome thermal noise generated by the heat of near sensor processing using a neural network trained on images with high amounts of thermal noise. This project is being carried out as an independent study which has started in the Spring of 2022.

Collaborators

Christopher Bruinsma and Yuhao Zhu at Horizon Research, *Univerisity of Rochester*

Image Capture

Images will be captured using a Flir BlackFly USB3 camera which has thermal-noise induced using a heat gun from Wagner. The safety of heating the camera ensured using Python code which relies on the Spinnaker SDK to moniter the camera temperature which is limited by the camera to less than 100°C. Having now pushed the camera to 100°C it is evident that the camera has a safety shutoff at this temperature.

This script is called `HeatTrigger.py`, it is essentially a camera trigger that captures at a specified temperature.

Depends on

```
import PySpin
import sys
import time
```

Runs as

```
$ Python3 HeatTrigger.py
```

Main additions to FLIR SDK example: `Trigger.py` are :

For the device temperature: `GetCameraTemperature(cam)` :

```
def GetCameraTemperature(cam):
    x = 0
    if cam.DeviceTemperature.GetAccessMode() == PySpin.RO:
        x = cam.DeviceTemperature.ToString()
    x = float(x)
    return x
```

as well as: `Go(cam,GoalTemperature)`

```
def Go(cam, GoalTemperature):
    # Get Temperature of Camera

    Temp = GetCameraTemperature(cam)
    print(GoalTemperature)

    # Heating
    while Temp < GoalTemperature:
        cam.Init()
        Temp = GetCameraTemperature(cam)
        print(Temp)
        time.sleep(3)  # Protects the camera.

    # Capture 2 images
    if Temp > GoalTemperature:
        # cam.DeInit() This makes the who thing crash much more quickly
        print("Capturing, please continue heating")
        Capture(cam)  # Cites : FLIR TELEDYNE
```

The heat testing is done using a loop in the `main()` method.

```
def main():
    •••
    # List of Cameras
    for i, cam in enumerate(cam_list):
        # List of Temperatures
        for t in range(60, 95, 1): # Capture more images per cycle
            # Initiates Capture
            Go(cam, t)
            time.sleep(2)

    print("Capture Complete, please cool the camera.")
    •••
```

Images are saved as `sample-serialNumber-capNum-temp.png`. These are RAW image files encoded as .png files.

The numbering relies on the `CamConfig.json` which stores the number of captures after each capture. In the terminal it looks like the following

```
•••
$ Acquiring images...
$ Image saved at sample-18255214-12-39.png

$ Image saved at sample-18255214-13-39.png

$ Trigger mode disabled...
•••
```

Machine Learning

Due to the nature of image processing of noisy images, max-pooling will likely be used alongside some kind of edge dectection algorthim. This aspect very much remains in the research stage, but as of right now the goal is to train a Convolution Neural Network to indentify cups of coffee that are either hot or iced. This implementation relies on TensorFlow.Keras.

This machine-learning model relies heavily on `2DConvolutions` and `Batch Normalizations`. However, at present it is currently not very effective, the goal now is to add more `maxPooling` to the model.

Runs as

```
$ IgnoresThermal.py
```

Intent

Indentify cups of coffee as either iced or hot. This will be done using a variety of coffee cups from the on-campus Starbucks here at the Univeristy that contain hot or iced coffee. These are contained within the `Training_Data` folder.

Depends on

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

As well as `pydot` ***and*** `graphviz`.

The data used is from the `training_images` folder which contains 599+ images taken by the **Blackfly** camera of cofee cupts of iced or hot varieties taken at various temperatures ranging from 60-97°C. These images are divided into two classes `Hot` and `Iced` and futher into `training` and `validation` within their respective folders.

The Code

I have done the following to create a neural network that uses data augmentation to virtually increase the sample size, as well as varying sized convolution kernels, batch normalization, making more dense the layers of the network and finally dropping layers out at each iteration to help train the network of more key characteristics.

```
def NModel(input_shape, num_classes):
    inputs = keras.Input(shape=input_shape)
    # Image augmentation block
    x = data_augmentation(inputs)
    x = layers.add([x, x])  # Add back residual
    x = layers.Rescaling(1.0 / 255)(x)
```

```
    # Entry block
    x = layers.Conv2D(1, 1, strides=2, padding="same")(x)
    x = layers.BatchNormalization()(x)
    x = layers.Dense(1, activation="softmax")(x)
    x = layers.add([x, x])  # Add back residual
    x = layers.Dropout(0.1)(x)

    x = layers.Conv2D(2, 1, strides=2, padding="same")(x)
    x = layers.add([x, x])  # Add back residual
    x = layers.BatchNormalization()(x)
    x = layers.Dense(1, activation="softmax")(x)
    x = layers.Dropout(0.2)(x)

    x = layers.Conv2D(4, 1, strides=2, padding="same")(x)
    x = layers.BatchNormalization()(x)
    x = layers.Dense(1, activation="softmax")(x)
    x = layers.Dropout(0.3)(x)

    x = layers.Conv2D(16, 1, strides=2, padding="same")(x)
    x = layers.BatchNormalization()(x)
    x = layers.Dense(1, activation="softmax")(x)
    x = layers.Dropout(0.4)(x)

    previous_block_activation = x  # Set aside residual

    for size in [1, 2, 4, 16]:
        x = layers.Activation("relu")(x)
        x = layers.SeparableConv2D(size, 3, padding="same")(x)
        x = layers.BatchNormalization()(x)

        x = layers.MaxPooling2D(3, strides=2, padding="same")(x)
        # Project residual
        residual = layers.Conv2D(size, 1, strides=2, padding="same")(
            previous_block_activation
        )
        x = layers.add([x, residual])  # Add back residual
        previous_block_activation = x  # Set aside next residual


    x = layers.GlobalAveragePooling2D()(x)
    x = layers.Dense(1, activation="softmax")(x)
    if num_classes == 2:
        activation = "sigmoid"
        units = 1
    else:
        activation = "softmax"
        units = num_classes

    x = layers.Dropout(0.1)(x)
    outputs = layers.Dense(units, activation=activation)(x)
    return keras.Model(inputs, outputs)
```

When Running

```
(venv) chris@dhcp-10-5-48-92 CSC_Independent % Python3 ignoresthermal.py
Found 465 files belonging to 2 classes.
Using 372 files for training.
2022-02-28 22:10:32.812495: I
tensorflow/core/platform/cpu_feature_guard.cc:151]
This TensorFlow binary is optimized with oneAPI Deep Neural Network
Library (oneDNN)
to use the following CPU instructions in performance-critical operations:
AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the
appropriate compiler flags.
Found 465 files belonging to 2 classes.
Using 93 files for validation.
Please Enter 1 for compile 2 for test and 3 for both
3

•••
Epoch 1/50
75/75 [==============================] - 69s 882ms/step - loss: 0.5198 -
accuracy: 0.7903 - val_loss: 0.4419 - val_accuracy: 0.8387
•••
Epoch 36/50
75/75 [==============================] - 71s 942ms/step - loss: 0.0949 -
accuracy: 0.9704 - val_loss: 1.6228 - val_accuracy: 0.6237
•••
Epoch 45/50
75/75 [==============================] - 78s 1s/step - loss: 0.0243 -
accuracy: 0.9919 - val_loss: 0.0244 - val_accuracy: 1.0000
Epoch 46/50
75/75 [==============================] - 76s 1s/step - loss: 0.1277 -
accuracy: 0.9597 - val_loss: 0.0411 - val_accuracy: 0.9785
Epoch 47/50
75/75 [==============================] - 75s 1s/step - loss: 0.0803 -
accuracy: 0.9812 - val_loss: 0.1400 - val_accuracy: 0.9355
Epoch 48/50
75/75 [==============================] - 72s 961ms/step - loss: 0.0573 -
accuracy: 0.9839 - val_loss: 0.0492 - val_accuracy: 0.9785
Epoch 49/50
75/75 [==============================] - 72s 957ms/step - loss: 0.1275 -
accuracy: 0.9462 - val_loss: 0.1103 - val_accuracy: 0.9785
Epoch 50/50
75/75 [==============================] - 74s 983ms/step - loss: 0.0695 -
accuracy: 0.9812 - val_loss: 0.0585 - val_accuracy: 0.9892

This image is 50.04 percent hot coffee and this image is 49.96 percent
iced coffee.
This image is 50.04 percent hot coffee and this image is 49.96 percent
iced coffee.
•••
```

Example Tests | Perhaps it just isn't quite sure which is which

```
Please Enter 1 for compile 2 for test and 3 for both
2
Images of hot coffee
This image is 50.01 percent hot coffee and this image is 49.99 percent
iced coffee.
This image is 50.01 percent hot coffee and this image is 49.99 percent
iced coffee.
Images of iced coffee
This image is 50.01 percent hot coffee and this image is 49.99 percent
iced coffee.
This image is 50.03 percent hot coffee and this image is 49.97 percent
iced coffee.
```

Analysis

Noisy Images

The first image is *hot coffee*, and the second image is *iced coffee*. At present, the levels of `accuracy` and `val_accuracy` are high. There are also at present two classes of image, those that are iced coffees and those that are hot coffees both taken with high levels of induced thermal noise.

Clearly the classification was *incorrect* but at currently there are `391` images of hot coffee and `212` of iced coffee. Each using `72` and `36` images for validation following a *80-20* training-to-validation rule.

The difficulty the model has in predicting where a coffee is iced or hot is likely because there are too few images at present to truly train the network on. More to come.

Denoised Images

These images are created by taking the RAW-encoded.png files and then denoising them using `Adobe Lightroom`

Works Cited :

> Dynamic Temperature Management of Near-Sensor Processing for Energy-Efficient High-Fidelity Imaging. Kodukula Et Al.

> Dirty Pixels: Towards End-to-End Image Processing and Perception Diamond Et. Al.

> FLIR. (n.d.). Spinnaker-SDKVersion (Trigger.py). Spinnaker SDK. Retrieved from https://www.flir.com/products/spinnaker-sdk/.

> FLIR Integrated Imaging Solutions, Inc. (n.d.). PySpinDoc.

> François Chollet, Team, K. (n.d.). Keras documentation: Image Classification From Scratch. Keras. Retrieved February 22, 2022, from https://keras.io/examples/vision/image_classification_from_scratch/