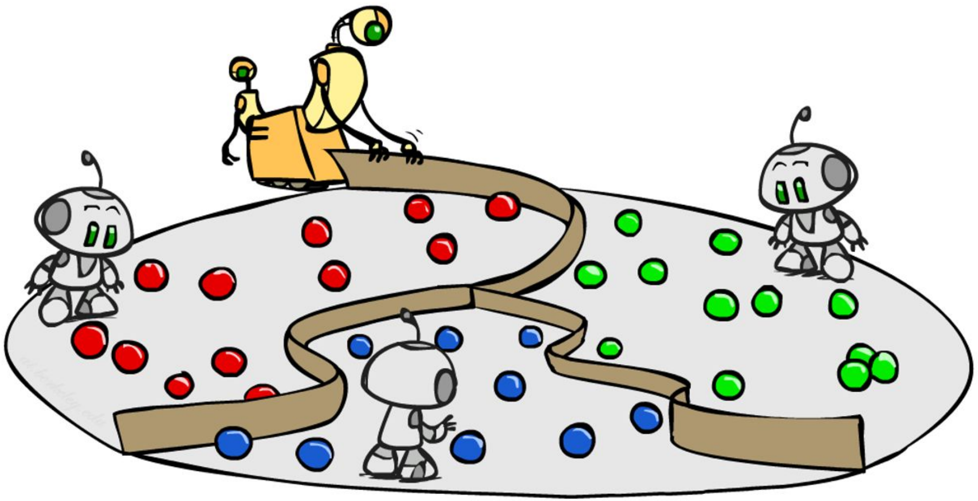
CS-ELEC2C: Machine Learning

Lab Exercise #3: What's Their Hair Type?



Exploring CNNs Through Image Classification

Dataset: Download it from the Lab Exercise #3 Module in Canvas

Task: You have a dataset containing hair images with 3 different types: **Wavy**, **Curly**, and **Straight** Hair. Your task is to **create a CNN** that can **classify** whether or not a person has wavy, curly, and straight hair.

Requirements:

1. Preprocess the Data
2. Create a Train and Test Split
3. Create a Convolutional Neural Network using Keras
4. Experiment on Various Elements
5. Discussion and Analysis of Experiments

Submission:

1. Code
2. Write-up of Activity (from Step #1 to Step #5)
    a. Must be in 2-column format (either in MS Word or LaTeX
    b. File must be in PDF

Part 1: Checking the Dataset

Image Checking: Check if all files are valid

<u>Please make sure that the hair_types directory is in the same location as your Jupyter Notebook, else this entire lab exercise won't work.</u>

```
from pathlib import Path
import imghdr
import os

data_dir = "hair_types"
image_extensions = [".png", ".jpg"]  # add there all your images file extensions

img_type_accepted_by_tf = ["bmp", "gif", "jpeg", "png"]
for filepath in Path(data_dir).rglob("*"):
    if filepath.suffix.lower() in image_extensions:
        img_type = imghdr.what(filepath)
        if img_type is None:
            print(f"{filepath} is not an image")
            os.remove(filepath)
        elif img_type not in img_type_accepted_by_tf:
            print(f"{filepath} is a {img_type}, not accepted by TensorFlow")
            os.remove(filepath)
```

Part 2: Using a DataLoader for Data Loading

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

image_size = (64, 64)
batch_size = 32

train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    "hair_types",
    validation_split=0.2,
    subset="training",
    seed=1337,
    image_size=image_size,
    batch_size=batch_size,
    labels='inferred',
    label_mode='categorical'
)
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    "hair_types",
    validation_split=0.2,
    subset="validation",
    seed=1337,
    image_size=image_size,
    batch_size=batch_size,
    labels='inferred',
    label_mode='categorical'
)
```

This is the code for the DataLoader. This allows us to **load the images from the directory itself** without having to load it in memory immediately.

The important parameters here is the **image size** (so if this is specified, the loader automatically resizes the image), the **batch size** (which is the number of images per iteration in each epoch), and **validation split**, and **seed**. Making the *train_ds* and *val_ds* have the same seed means ***that the dataset is split the same***.

Please check this API for more details: https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image_dataset_from_directory

Part 3: Visualizing the Data

```
import matplotlib.pyplot as plt
import numpy as np

plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(int(np.argmax(labels[i])))
        plt.axis("off")
plt.show()
```

Part 4: Sample Model

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()
model.add(keras.Input(shape=image_size + (3,))) # 64, 64, 3
model.add(layers.Rescaling(1.0 / 255))

model.add(layers.Conv2D(filters=4, kernel_size=16, strides=1, padding='valid', dilation_rate=1))
model.add(layers.Activation("relu"))

model.add(layers.Conv2D(filters=8, kernel_size=8, strides=1, padding='valid', dilation_rate=1))
model.add(layers.Activation("relu"))

model.add(layers.Conv2D(filters=16, kernel_size=4, strides=1, padding='valid', dilation_rate=1))
model.add(layers.Activation("relu"))

model.add(layers.GlobalAveragePooling2D())
model.add(layers.Dense(64))
model.add(layers.Activation("relu"))
model.add(layers.Dense(3))
model.add(layers.Activation("softmax"))

tf.keras.utils.plot_model(model, to_file='model_test.png', show_shapes=True)

epochs = 50

model.compile(
    optimizer=keras.optimizers.Adam(1e-3),
    loss="categorical_crossentropy",
    metrics=["accuracy"],
)
model.fit(train_ds, epochs=epochs, validation_data=val_ds)
```

Keras is an easy-to-use interface for creating neural networks. It can be seen that it is mostly used in this lab exercise. The backend of Keras is Tensorflow, meaning its backend computations is done through the Tensorflow library

In this code, we initialize the model as a ***Sequential* model**. Meaning, the **layers are supposed to be connected in sequence**. ***Advanced applications don't always follow a sequence.***

The first set of layers are mostly trying to resize and rescale the image. The **normalization was added as a part of the layer**. Furthermore, you can see **Conv2D layers with different hyperparameters, signifying the different convolutions** and a GlobalAveragePooling. **GlobalAveragePooling computes the average pooling across channels** rather than across the feature space.

Part 4: Sample Model

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()
model.add(keras.Input(shape=image_size + (3,))) # 64, 64, 3
model.add(layers.Rescaling(1.0 / 255))

model.add(layers.Conv2D(filters=4, kernel_size=16, strides=1, padding='valid', dilation_rate=1))
model.add(layers.Activation("relu"))

model.add(layers.Conv2D(filters=8, kernel_size=8, strides=1, padding='valid', dilation_rate=1))
model.add(layers.Activation("relu"))

model.add(layers.Conv2D(filters=16, kernel_size=4, strides=1, padding='valid', dilation_rate=1))
model.add(layers.Activation("relu"))

model.add(layers.GlobalAveragePooling2D())
model.add(layers.Dense(64))
model.add(layers.Activation("relu"))
model.add(layers.Dense(3))
model.add(layers.Activation("softmax"))

tf.keras.utils.plot_model(model, to_file='model_test.png', show_shapes=True)

epochs = 50

model.compile(
    optimizer=keras.optimizers.Adam(1e-3),
    loss="categorical_crossentropy",
    metrics=["accuracy"],
)
model.fit(train_ds, epochs=epochs, validation_data=val_ds)
```

After the Global Average Pooling, you can see that the **next layer is a set of Dense Layers, these are your typical hidden layers** that we've discussed in the previous lectures. This is where the classification happens.

Afterwards, the **neural network architecture is plot using the *plot_model* code** and you can see it in your directory as the *model_test.png* file.

Afterwards, the ***epochs*** denote **how many times we want the neural network to pass through the entire dataset**.

Next, the model is compiled where the **optimizer is an Adam Optimizer with a learning rate of 1e-3**. The Adam optimizer is a variant of the gradient descent that we learned. The loss is categorical cross entropy and we're tracking accuracy.

Part 5: Predictions for Images

```
img = keras.preprocessing.image.load_img(

"hair_types/Curly_Hair/02dac897d1dec9ba8c057a11d041ada8--layered-natural-hair-natural-black-hai
rstyles.jpg", target_size=image_size
)
img_array = keras.preprocessing.image.img_to_array(img)
img_array = tf.expand_dims(img_array, 0)  # Create batch axis

predictions = model.predict(img_array)
print(
    "This image is %.2f percent curly hair, %.2f percent straight hair, and %.2f percent wavy
hair."
    % tuple(predictions[0])
)
```

What You Need To Do

Possible Things To Experiment On:

- Other Preprocessing Methods for Images
- Adding Max Pooling
- Changing Number of Filters
- Changing Kernel Size
- Changing Learning Rate
- Changing Optimizers, etc…

For the Write-Up:

- Tell Story from Loading the Data to Insights
- Insights on Experiments:
    - Effects on Changing ____ with Respect to Performance
    - Effects of Doing ____ on Qualitative Errors