NFT_Classification

August 8, 2022

1 NFT Image Classification

1.0.1 Convolution Neural Network

by: Tyler Cranmer

Non-fungible tokens, or NFTs, are pieces of digital content linked to the blockchain, the digital database underpinning cryptocurrencies such as bitcoin and ethereum. Unlike NFTs, those assets are fungible, meaning they can be replaced or exchanged with another identical one of the same value, much like a dollar bill.

NFTs, on the other hand are unique and not mutally interchangeable, which means no two NFTs are the same. Think of them as rare baseball trading cards or pokemon trading cards. Just like there are a collection of branded Basketball trading cards or Yugioh playing cards, NFTs are typically made in different "Collections". Some of the most popular NFT Collections include the Bored Ape Yatch Club, Moonbirds, Crypto Punks and Goblin Town.

The goal is this project is to utilize Convolution Neural Network to classify different NFT pictures into their respected collections. This project will utilize 6000 pictures from 6 different NFT collections to train and test the model.

2 Data Explanation

The data used in the project was scraped from varies IPFS servers where the NFT metadata was hosted. 6000 unique NFT Pictures from 6 Collections.

Collections: - Bored Ape Yatch Club - Doodle - Goblin Town - Moonbird - Prime Ape Yatch Club

Files:

train - bayc (800 bored ape yatch club NFT pictures) - doodle (800 doodle NFT pictures) - GoblinTown (800 goblin town NFT pictures) - JustApe (800 Just Ape NFT pictures) - Moonbird (800 moonbird NFT pictures) - PAYC (800 prime ape yatch club NFT pictures)

test - bayc (200 bored ape yatch club NFT pictures) - doodle (200 doodle NFT pictures) - GoblinTown (200 goblin town NFT pictures) - JustApe (200 Just Ape NFT pictures) - Moonbird (200 moonbird NFT pictures) - PAYC (200 prime ape yatch club NFT pictures)

```
[2]: import numpy as np
     import matplotlib.pyplot as plt
     import matplotlib.image as mpimg
     # import cv2
     import pandas as pd
     import os
     from PIL import Image

     import tensorflow as tf
     import keras_tuner
     from tensorflow import keras
     from keras_tuner.tuners import RandomSearch
     from tensorflow.keras.models import Sequential
     from tensorflow.keras.utils import image_dataset_from_directory
     from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten,␣
      ↪Rescaling
```

[3]:
```
os.environ["KERAS_BACKEND"] = "plaidml.keras.backend"
```

3 Data Processing

3.0.1 Loading Dataset

[4]:
```
test_dir = './test'
train_dir = './train'

train_dir_doodle = train_dir + '/doodle'
train_dir_bayc = train_dir + '/bayc'
train_dir_goblin = train_dir + '/GoblinTown'
train_dir_justape = train_dir + '/JustApe'
train_dir_moonbird = train_dir + '/Moonbird'
train_dir_payc = train_dir + '/PAYC'

test_dir_doodle = test_dir + '/doodle'
test_dir_bayc = test_dir + '/bayc'
test_dir_goblin = test_dir + '/GoblinTown'
test_dir_justape = test_dir + '/JustApe'
test_dir_moonbird = test_dir + '/Moonbird'
test_dir_payc = test_dir + '/PAYC'
```

3.0.2 Dataset Info

We have 1000 NFT Images from each collection that will be split into a training and testing set. The training set will have 80% of the pictures and the test set will contain 20% of the total collection.

[5]:
```
files = [('Doodle', train_dir_doodle, test_dir_doodle), ('Bored Ape Yatch␣
 ↪Club', train_dir_bayc, test_dir_bayc),
         ('Prime Ape Yatch Club', train_dir_payc, test_dir_payc),
        ('Just Ape', train_dir_justape, test_dir_justape), ('Goblin Town',␣
↪train_dir_goblin, test_dir_goblin),
         ('MoonBird', train_dir_moonbird, test_dir_moonbird) ]
for data in files:
    print(f'Number of {data[0]} NFT training images - {len(os.
↪listdir(data[1]))}')
    print(f'Number of {data[0]} NFT training images - {len(os.
↪listdir(data[2]))} \n')
```

```
Number of Doodle NFT training images - 800
Number of Doodle NFT training images - 200

Number of Bored Ape Yatch Club NFT training images - 800
Number of Bored Ape Yatch Club NFT training images - 200

Number of Prime Ape Yatch Club NFT training images - 800
Number of Prime Ape Yatch Club NFT training images - 200

Number of Just Ape NFT training images - 800
Number of Just Ape NFT training images - 200

Number of Goblin Town NFT training images - 800
Number of Goblin Town NFT training images - 200

Number of MoonBird NFT training images - 800
Number of MoonBird NFT training images - 200
```

[ ]:

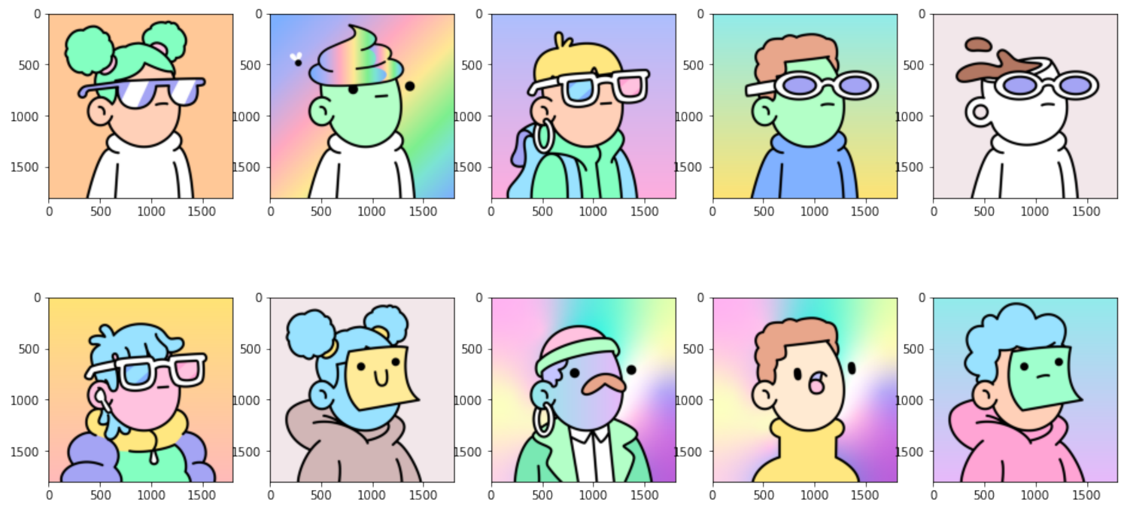
3.0.3 Visualization of Pictures

[6]:
```
def viz_pics(path):
    paths = []
    for root, directories, files in os.walk(path):
        for i, filename in enumerate(files):
            if i < 10:
                filepath = os.path.join(root, filename)
                paths.append(filepath)

    plt.figure(figsize=(16,8))

    for i, picture in enumerate(paths, start=1):
        plt.subplot(2,5,i)
        image = mpimg.imread(picture)
        print(f'shape of the image {i} : {image.shape}')
        plt.imshow(image)
    plt.show()
```

3.0.4 Doodle NFT

[7]:
```
viz_pics(files[0][1])
```

```
shape of the image 1 : (1800, 1800, 4)
shape of the image 2 : (1800, 1800, 4)
shape of the image 3 : (1800, 1800, 4)
shape of the image 4 : (1800, 1800, 4)
shape of the image 5 : (1800, 1800, 4)
shape of the image 6 : (1800, 1800, 4)
shape of the image 7 : (1800, 1800, 4)
shape of the image 8 : (1800, 1800, 4)
shape of the image 9 : (1800, 1800, 4)
shape of the image 10 : (1800, 1800, 4)
```

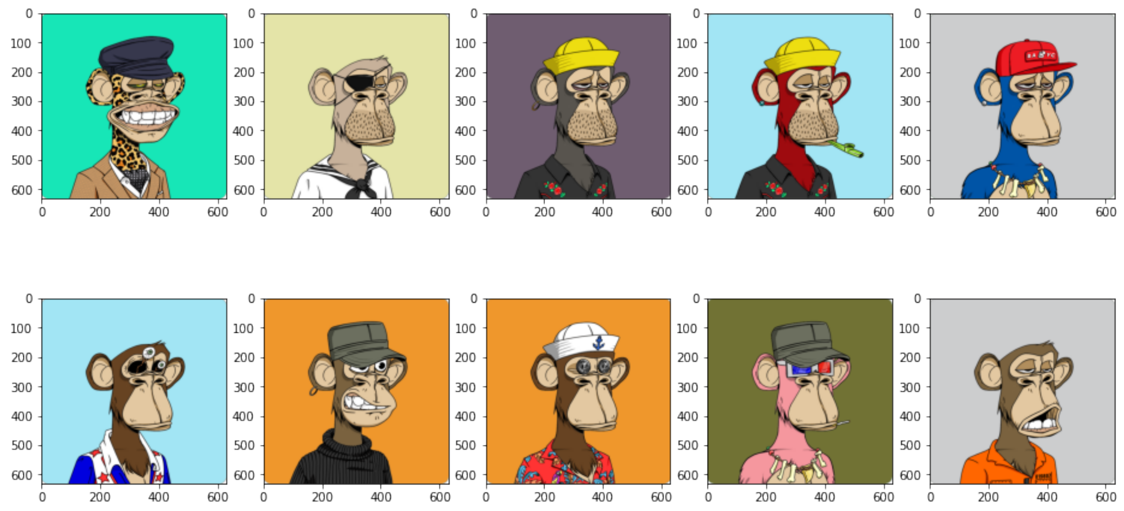


[ ]:
```

```

3.0.5 Bored Ape Yatch Club NFT

[8]:
```
viz_pics(files[1][1])
```

```
shape of the image 1 : (631, 631, 4)
shape of the image 2 : (631, 631, 4)
shape of the image 3 : (631, 631, 4)
shape of the image 4 : (631, 631, 4)
shape of the image 5 : (631, 631, 4)
shape of the image 6 : (631, 631, 4)
shape of the image 7 : (631, 631, 4)
shape of the image 8 : (631, 631, 4)
shape of the image 9 : (631, 631, 4)
shape of the image 10 : (631, 631, 4)
```



3.0.6 Prime Ape Yatch Club NFT

```
[9]: viz_pics(files[2][1])
```

```
shape of the image 1 : (1500, 1500, 3)
shape of the image 2 : (1500, 1500, 3)
shape of the image 3 : (1500, 1500, 3)
shape of the image 4 : (1500, 1500, 3)
shape of the image 5 : (1500, 1500, 3)
shape of the image 6 : (1500, 1500, 3)
shape of the image 7 : (1500, 1500, 3)
shape of the image 8 : (1500, 1500, 3)
shape of the image 9 : (1500, 1500, 3)
shape of the image 10 : (1500, 1500, 3)
```

3.0.7 Just Ape NFT

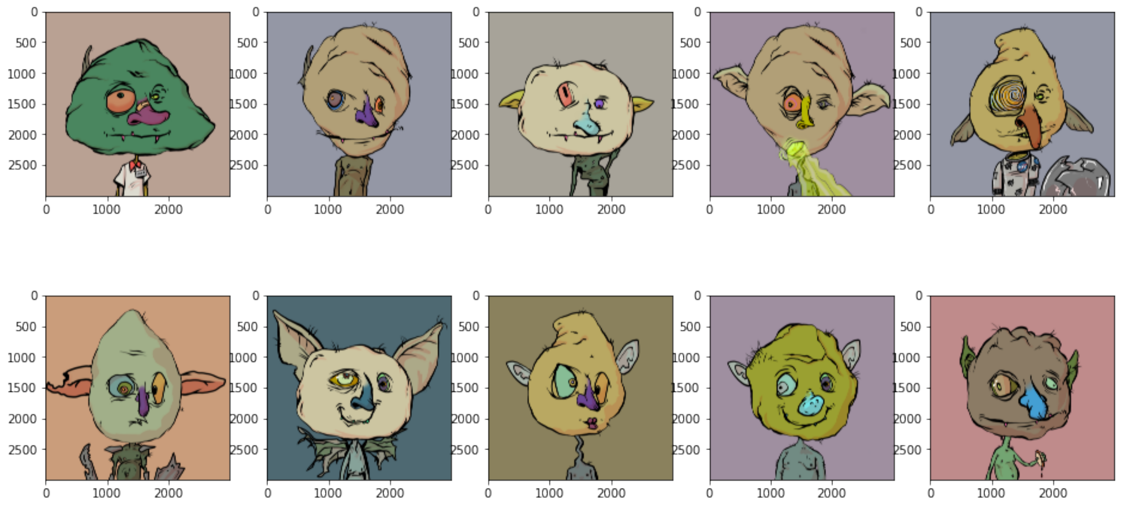
```
[10]: viz_pics(files[3][1])
```

```
shape of the image 1 : (1200, 1200, 4)
shape of the image 2 : (1200, 1200, 4)
shape of the image 3 : (1200, 1200, 4)
shape of the image 4 : (1200, 1200, 4)
shape of the image 5 : (1200, 1200, 4)
shape of the image 6 : (1200, 1200, 4)
shape of the image 7 : (1200, 1200, 4)
shape of the image 8 : (1200, 1200, 4)
shape of the image 9 : (1200, 1200, 4)
shape of the image 10 : (1200, 1200, 4)
```

3.0.8 Goblin Town NFT

[11]:
```
viz_pics(files[4][1])
```

```
shape of the image 1 : (3000, 3000, 4)
shape of the image 2 : (3000, 3000, 4)
shape of the image 3 : (3000, 3000, 4)
shape of the image 4 : (3000, 3000, 4)
shape of the image 5 : (3000, 3000, 4)
shape of the image 6 : (3000, 3000, 4)
shape of the image 7 : (3000, 3000, 4)
shape of the image 8 : (3000, 3000, 4)
shape of the image 9 : (3000, 3000, 4)
shape of the image 10 : (3000, 3000, 4)
```



3.0.9 Moonbird NFT

[12]:
```
viz_pics(files[5][1])
```

```
shape of the image 1 : (1008, 1008, 3)
shape of the image 2 : (1008, 1008, 3)
shape of the image 3 : (1008, 1008, 3)
shape of the image 4 : (1008, 1008, 3)
shape of the image 5 : (1008, 1008, 3)
shape of the image 6 : (1008, 1008, 3)
shape of the image 7 : (1008, 1008, 3)
shape of the image 8 : (1008, 1008, 3)
shape of the image 9 : (1008, 1008, 3)
shape of the image 10 : (1008, 1008, 3)
```

As we can see from the data, each NFT collection has different dimensions. There is a range of sizes with each collection, where the Bored Ape Yatch Club Collection has the smallest picture size of 631 X 631 pixels and The Globin Town NFT collection being the size of 3000 x 3000 pixels. There is a range between the different color channels each picture is using. Some of the collections use 3 (RGB) or 4 (CMYK) values. We will have to resize each image to be the same dimension for our CNN to work properly.

3.1 Creating Train and Test Data

We are going to split the data into to sets, training data and testing data that consists of images from each collection. The images will all be resized to be of shape 180px by 180px. This will help the CNN process the data quicker, since the amount of data will be shrunk down to a smaller dimension. We will also transform the images to all have a color channel of 3 (RGB).

[13]:
```
batch_size = 32
img_height = 180
img_width = 180

train_ds = image_dataset_from_directory(train_dir, image_size=(img_height,␣
 ↪img_width), batch_size=batch_size)
```

```
Found 4800 files belonging to 6 classes.
```

[14]:
```
test_ds = image_dataset_from_directory(test_dir, image_size=(img_height,␣
 ↪img_width), batch_size=batch_size)
```

```
Found 1200 files belonging to 6 classes.
```

[15]:
```
class_names = train_ds.class_names
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(class_names[labels[i]])
        plt.axis("off")
```

JustApe
Moonbird
bayc
GoblinTown
bayc
Moonbird
GoblinTown
bayc
JustApe

3.1.1 Batch Images

Image Batch is a tensor of the shape (32, 180,180,3). Meaning, this batch consist of 32 images of shape (180x180x3). The last dimension refers to color channels RGB.

[16]:
```
for image_batch, labels_batch in train_ds:
    print(image_batch.shape)
    break
```

```
(32, 180, 180, 3)
```

3.1.2 Standardizing Image Data

RGB channel values are between (0,255). This is not ideal for a neural network, gernally we want our input values to be small. I am going to standardize values to be between (0,1) https://www.tensorflow.org/tutorials/load_data/images

[17]:
```
normalization_layer = Rescaling(1./255)
normalized_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
image_batch, labels_batch = next(iter(normalized_ds))
first_image = image_batch[0]
# The pixel values are now in `[0,1]`.
print(np.min(first_image), np.max(first_image))
```

```
0.0 1.0
```

3.1.3 Configuring dataset performance

Let's make sure to use buffered prefetching so you can yield data from disk without having I/O become blocking. These are two important methods you should use when loading data:

Dataset.cache keeps the images in memory after they're loaded off disk during the first epoch. This will ensure the dataset does not become a bottleneck while training your model. If your dataset is too large to fit into memory, you can also use this method to create a performant on-disk cache.

Dataset.prefetch overlaps data preprocessing and model execution while training.

[18]:
```
AUTOTUNE = tf.data.AUTOTUNE

train_ds = train_ds.cache().prefetch(buffer_size=AUTOTUNE)
val_ds = test_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

Now that our image data has been preprocessed and normalized, we are ready to start building the CNN models.

4 Modeling

We will start with a simple CNN network and build complexity into future models

- Input Layer: It represent input image data. It will reshape image into single diminsion array. Example your image is 180x180 = 32400, it will convert to (32400,3) array.
- Conv Layer: This layer will extract features from image.
- Pooling Layer: This layerreduce the spatial volume of input image after convolution.
- Fully Connected Layer: It connect the network from a layer to another layer Output Layer: It is the predicted values layer

4.0.1 Base Model (Model 1)

We will start with a model that has 1 convolution and max pooling layer that is connected a fully connected network of 64 nodes

[19]:
```
num_classes = 6

model = tf.keras.Sequential([
  tf.keras.layers.Rescaling(1./255),
  tf.keras.layers.Conv2D(32, 3, activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(64, activation='softmax'),
  tf.keras.layers.Dense(num_classes)
])
model.compile(
optimizer='adam', loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=True), metrics=['accuracy'])
model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=3
)
```

```
Epoch 1/3
150/150 [==============================] - 60s 383ms/step - loss: 1.8000 -
accuracy: 0.1702 - val_loss: 1.7965 - val_accuracy: 0.1667
Epoch 2/3
150/150 [==============================] - 41s 272ms/step - loss: 1.7946 -
accuracy: 0.1604 - val_loss: 1.7929 - val_accuracy: 0.1667
Epoch 3/3
150/150 [==============================] - 41s 270ms/step - loss: 1.7927 -
accuracy: 0.1606 - val_loss: 1.7920 - val_accuracy: 0.1667
```

[19]:
```
<keras.callbacks.History at 0x1440ae850>
```

[20]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_1 (Rescaling)     (None, 180, 180, 3)       0

 conv2d (Conv2D)             (None, 178, 178, 32)      896

 max_pooling2d (MaxPooling2D  (None, 89, 89, 32)       0
 )
 flatten (Flatten)           (None, 253472)           0

 dense (Dense)               (None, 64)               16222272

 dense_1 (Dense)             (None, 6)                390

=================================================================
Total params: 16,223,558
Trainable params: 16,223,558
Non-trainable params: 0
_________________________________________________________________
```

4.0.2 Model 2

For this model, we will increase the number of convolution layers and maxpooling by 1. This will add more complexity to the network.

[21]:
```python
model_2 = tf.keras.Sequential([
  tf.keras.layers.Rescaling(1./255),
  tf.keras.layers.Conv2D(32, 3, activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Conv2D(32, 3, activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(64, activation='softmax'),
  tf.keras.layers.Dense(num_classes)
])

model_2.compile(
optimizer='adam', loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=True), metrics=['accuracy'])
model_2.fit(
  train_ds,
  validation_data=val_ds,
  epochs=3
)
```

```
Epoch 1/3
150/150 [==============================] - 55s 368ms/step - loss: 1.5344 -
accuracy: 0.6137 - val_loss: 1.4281 - val_accuracy: 0.6650
Epoch 2/3
150/150 [==============================] - 56s 373ms/step - loss: 1.3492 -
accuracy: 0.6658 - val_loss: 1.2744 - val_accuracy: 0.6650
Epoch 3/3
150/150 [==============================] - 55s 368ms/step - loss: 1.2091 -
accuracy: 0.6658 - val_loss: 1.1467 - val_accuracy: 0.6658
```

[21]:
```
<keras.callbacks.History at 0x143fa4970>
```

[22]:
```
model_2.summary()
```

```
Model: "sequential_1"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_2 (Rescaling)     (None, 180, 180, 3)       0

 conv2d_1 (Conv2D)           (None, 178, 178, 32)      896

 max_pooling2d_1 (MaxPooling  (None, 89, 89, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 87, 87, 32)        9248

 max_pooling2d_2 (MaxPooling  (None, 43, 43, 32)       0
 2D)

 flatten_1 (Flatten)         (None, 59168)             0

 dense_2 (Dense)             (None, 64)                3786816

 dense_3 (Dense)             (None, 6)                 390

=================================================================
Total params: 3,797,350
Trainable params: 3,797,350
Non-trainable params: 0
_________________________________________________________________
```

4.0.3 Model 3

This model will again increase the number of convolution and maxpooling. We will also increase the number of epocs to 5.

[23]:
```
model_3 = tf.keras.Sequential([
  tf.keras.layers.Rescaling(1./255),
  tf.keras.layers.Conv2D(32, 3, activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Conv2D(32, 3, activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Conv2D(32, 3, activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(64, activation='softmax'),
  tf.keras.layers.Dense(num_classes)
])
model_3.compile(
optimizer='adam', loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=True), metrics=['accuracy'])
model_3.fit(
  train_ds,
  validation_data=val_ds,
  epochs=5
)
```

```
Epoch 1/5
150/150 [==============================] - 62s 409ms/step - loss: 1.5143 -
accuracy: 0.7733 - val_loss: 1.3904 - val_accuracy: 0.8300
Epoch 2/5
150/150 [==============================] - 60s 403ms/step - loss: 1.2991 -
accuracy: 0.8300 - val_loss: 1.2110 - val_accuracy: 0.8317
Epoch 3/5
150/150 [==============================] - 59s 390ms/step - loss: 1.1352 -
accuracy: 0.8319 - val_loss: 1.0621 - val_accuracy: 0.8317
Epoch 4/5
150/150 [==============================] - 58s 388ms/step - loss: 0.9988 -
accuracy: 0.8315 - val_loss: 0.9354 - val_accuracy: 0.8325
Epoch 5/5
150/150 [==============================] - 58s 389ms/step - loss: 0.8806 -
accuracy: 0.8329 - val_loss: 0.8302 - val_accuracy: 0.8317
```

[23]:
```
<keras.callbacks.History at 0x142b19f10>
```

[24]:
```
model_3.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_3 (Rescaling)     (None, 180, 180, 3)       0

 conv2d_3 (Conv2D)           (None, 178, 178, 32)      896

 max_pooling2d_3 (MaxPooling  (None, 89, 89, 32)       0
 2D)

 conv2d_4 (Conv2D)           (None, 87, 87, 32)        9248

 max_pooling2d_4 (MaxPooling  (None, 43, 43, 32)       0
 2D)

 conv2d_5 (Conv2D)           (None, 41, 41, 32)        9248

 max_pooling2d_5 (MaxPooling  (None, 20, 20, 32)       0
 2D)

 flatten_2 (Flatten)         (None, 12800)             0

 dense_4 (Dense)             (None, 64)                819264

 dense_5 (Dense)             (None, 6)                 390

=================================================================
Total params: 839,046
Trainable params: 839,046
Non-trainable params: 0
_________________________________________________________________
```

4.0.4 Model 4

We will continue to add convolution and maxpooling layers.

[25]:
```python
model_4 = tf.keras.Sequential([
  tf.keras.layers.Rescaling(1./255),
  tf.keras.layers.Conv2D(32, 3, activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Conv2D(32, 3, activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Conv2D(32, 3, activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(64, activation='softmax'),
  tf.keras.layers.Dense(num_classes)
])
model_4.compile(
optimizer='adam', loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=True), metrics=['accuracy'])
model_4.fit(
  train_ds,
  validation_data=val_ds,
  epochs=5
)
```

```
Epoch 1/5
150/150 [==============================] - 60s 395ms/step - loss: 1.5029 -
accuracy: 0.7477 - val_loss: 1.3654 - val_accuracy: 0.8292
Epoch 2/5
150/150 [==============================] - 58s 389ms/step - loss: 1.2755 -
accuracy: 0.8296 - val_loss: 1.1875 - val_accuracy: 0.8325
Epoch 3/5
150/150 [==============================] - 58s 389ms/step - loss: 1.1101 -
accuracy: 0.8329 - val_loss: 1.0375 - val_accuracy: 0.8325
```

```
Epoch 4/5
150/150 [==============================] - 59s 392ms/step - loss: 0.9743 -
accuracy: 0.8323 - val_loss: 0.9139 - val_accuracy: 0.8325
Epoch 5/5
150/150 [==============================] - 59s 395ms/step - loss: 0.8592 -
accuracy: 0.8333 - val_loss: 0.8095 - val_accuracy: 0.8333
```

[25]:
```
<keras.callbacks.History at 0x142c6dd60>
```

[26]:
```
model_4.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_4 (Rescaling)     (None, 180, 180, 3)       0

 conv2d_6 (Conv2D)           (None, 178, 178, 32)      896

 max_pooling2d_6 (MaxPooling  (None, 89, 89, 32)       0
 2D)

 conv2d_7 (Conv2D)           (None, 87, 87, 32)        9248

 max_pooling2d_7 (MaxPooling  (None, 43, 43, 32)       0
 2D)

 conv2d_8 (Conv2D)           (None, 41, 41, 32)        9248

 max_pooling2d_8 (MaxPooling  (None, 20, 20, 32)       0
 2D)

 flatten_3 (Flatten)         (None, 12800)             0

 dense_6 (Dense)             (None, 64)                819264

 dense_7 (Dense)             (None, 6)                 390

=================================================================
Total params: 839,046
Trainable params: 839,046
Non-trainable params: 0
_________________________________________________________________
```

4.0.5 Model 5

Model 5 will have some changed parameters to see if we can get the network to have an accuracy close to 100%. We doubled the number of nodes connected to the dense network at the end. Which increased the accuracy to 99.6%

[27]:
```
model_5 = tf.keras.Sequential([
  tf.keras.layers.Rescaling(1./255),
  tf.keras.layers.Conv2D(32, 3, activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Conv2D(32, 3, activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Conv2D(32, 3, activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(128, activation='softmax'),
  tf.keras.layers.Dense(num_classes)
])

model_5.compile(
optimizer='adam', loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=True), metrics=['accuracy'])
model_5.fit(
  train_ds,
  validation_data=val_ds,
  epochs=5
)
```

```
Epoch 1/5
150/150 [==============================] - 61s 405ms/step - loss: 1.5201 -
accuracy: 0.8025 - val_loss: 1.3925 - val_accuracy: 0.8300
Epoch 2/5
150/150 [==============================] - 61s 405ms/step - loss: 1.3033 -
accuracy: 0.8310 - val_loss: 1.2148 - val_accuracy: 0.8308
Epoch 3/5
150/150 [==============================] - 61s 406ms/step - loss: 1.1377 -
accuracy: 0.8321 - val_loss: 1.0666 - val_accuracy: 0.8300
Epoch 4/5
150/150 [==============================] - 60s 400ms/step - loss: 0.9989 -
accuracy: 0.8296 - val_loss: 0.9393 - val_accuracy: 0.8300
Epoch 5/5
150/150 [==============================] - 61s 406ms/step - loss: 0.8824 -
accuracy: 0.8294 - val_loss: 0.8334 - val_accuracy: 0.8300
```

[27]:
```
<keras.callbacks.History at 0x1443c27c0>
```

[28]:
```
model_5.summary()
```

```
Model: "sequential_4"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_5 (Rescaling)     (None, 180, 180, 3)       0

 conv2d_9 (Conv2D)           (None, 178, 178, 32)      896

 max_pooling2d_9 (MaxPooling  (None, 89, 89, 32)       0
 2D)

 conv2d_10 (Conv2D)          (None, 87, 87, 32)        9248

 max_pooling2d_10 (MaxPoolin  (None, 43, 43, 32)       0
 g2D)

 conv2d_11 (Conv2D)          (None, 41, 41, 32)        9248

 max_pooling2d_11 (MaxPoolin  (None, 20, 20, 32)       0
 g2D)

 flatten_4 (Flatten)         (None, 12800)             0

 dense_8 (Dense)             (None, 128)               1638528

 dense_9 (Dense)             (None, 6)                 774

=================================================================
Total params: 1,658,694
Trainable params: 1,658,694
Non-trainable params: 0
_________________________________________________________________
```

5 Analysis

The first model was used as a baseline to see if we can add more complexity to our Convolution Neural Networks. Since the training accuracy and test accuracy was well bellow an acceptable percentage, we knew that there was some room for more complexity within the network. To increase the performace of the network, we increased the number of epochs and added one more convolution and maxpooling layer to pull more features from the data. That did increase the accuracy, but there was room for more. After model 4, we had a very high accuracy and I wanted to see if we could get closer to 100% by increasing the complexity to the neural net at the end of the model. We doubled the number of nodes to 128, which increased the overall accuracy to 99.6%.

6 Discussion and Conclusion

The CNN models worked great as we added in more convolution layers and max pooling layers. The more filters we added, the stronger the models performed. I wanted to do dataset that was interesting to me and I chose the NFT set, not truly thinking about how simple the images were. Even the simple models had an easy time correctly classifying the right NFT to their collection. The simple pictures made it hard to fine tune the hyper parameters, because the models were already performing at near 100% accuracy. For next time, I would choose a different dataset that had more variation between the images.

[ ]:
```

```