```
import numpy as np
import pandas as pd
import seaborn as sns
import pathlib
import matplotlib.pyplot as plt
```

```
# Importing Tensorflow
import tensorflow as tf

from tensorflow import keras
from tensorflow.keras import layers

print(tf.__version__)
```

```
2.15.0
```

```
!pip install git+https://github.com/tensorflow/docs # Use some functions from tensorflow_docs
```

```
Collecting git+https://github.com/tensorflow/docs
  Cloning https://github.com/tensorflow/docs to /tmp/pip-req-build-yqtd1epg
  Running command git clone --filter=blob:none --quiet https://github.com/tensorflow/docs /tmp/pip-req-build-yqtd1epg
  Resolved https://github.com/tensorflow/docs to commit eea8c521822d188d136083d9af4c58e9c381b756
  Preparing metadata (setup.py) ... done
Collecting astor (from tensorflow-docs==2024.5.3.31743)
  Downloading astor-0.8.1-py2.py3-none-any.whl (27 kB)
Requirement already satisfied: absl-py in /usr/local/lib/python3.10/dist-packages (from tensorflow-docs==2024.5.3.31743) (1.4.0)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from tensorflow-docs==2024.5.3.31743) (3.1.4)
Requirement already satisfied: nbformat in /usr/local/lib/python3.10/dist-packages (from tensorflow-docs==2024.5.3.31743) (5.10.4)
Requirement already satisfied: protobuf>=3.12 in /usr/local/lib/python3.10/dist-packages (from tensorflow-docs==2024.5.3.31743) (3.20.3
Requirement already satisfied: pyyaml in /usr/local/lib/python3.10/dist-packages (from tensorflow-docs==2024.5.3.31743) (6.0.1)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->tensorflow-docs==2024.5.3.31743
Requirement already satisfied: fastjsonschema>=2.15 in /usr/local/lib/python3.10/dist-packages (from nbformat->tensorflow-docs==2024.5.
Requirement already satisfied: jsonschema>=2.6 in /usr/local/lib/python3.10/dist-packages (from nbformat->tensorflow-docs==2024.5.3.317
Requirement already satisfied: jupyter-core!=5.0.*,>=4.12 in /usr/local/lib/python3.10/dist-packages (from nbformat->tensorflow-docs==2
Requirement already satisfied: traitlets>=5.1 in /usr/local/lib/python3.10/dist-packages (from nbformat->tensorflow-docs==2024.5.3.3174
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6->nbformat->tensorflow-doc
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6->n
Requirement already satisfied: referencing>=0.28.4 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6->nbformat->tensorfl
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6->nbformat->tensorflow-do
Requirement already satisfied: platformdirs>=2.5 in /usr/local/lib/python3.10/dist-packages (from jupyter-core!=5.0.*,>=4.12->nbformat-
Building wheels for collected packages: tensorflow-docs
  Building wheel for tensorflow-docs (setup.py) ... done
  Created wheel for tensorflow-docs: filename=tensorflow_docs-2024.5.3.31743-py3-none-any.whl size=182531 sha256=d50d6615dc9063dc9c0f87
  Stored in directory: /tmp/pip-ephem-wheel-cache-f65hlbwc/wheels/86/0f/1e/3b62293c8ffd0fd5a49508e6871cdb7554abe9c62afd35ec53
Successfully built tensorflow-docs
Installing collected packages: astor, tensorflow-docs
Successfully installed astor-0.8.1 tensorflow-docs-2024.5.3.31743
```

```
import tensorflow_docs as tfdocs
import tensorflow_docs.plots
import tensorflow_docs.modeling
```

```
#plotly used for plotting Interactive Graphs
!pip install plotly
import plotly.offline as py
import plotly.express as px
import plotly.graph_objs as go
```

```
Requirement already satisfied: plotly in /usr/local/lib/python3.10/dist-packages (5.15.0)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from plotly) (8.3.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from plotly) (24.0)
```

Loading Dataset

```
dataset = pd.read_csv('final_data.csv')
dataset
```

	red	green	blue	label
0	20	139	240	Blue
1	174	83	72	Brown
2	144	249	131	Green
3	168	25	156	Pink
4	30	182	136	Green
...	...	...	...	...
5047	26	26	26	Black
5048	27	27	27	Black
5049	28	28	28	Black
5050	29	29	29	Black
5051	30	30	30	Black

5052 rows × 4 columns

Next steps: Generate code with dataset View recommended plots

```
dataset.describe()
```

	red	green	blue
count	5052.000000	5052.000000	5052.000000
mean	127.416073	126.316310	128.296912
std	73.958449	74.927131	74.174008
min	0.000000	0.000000	0.000000
25%	63.000000	60.000000	63.000000
50%	128.000000	127.000000	130.000000
75%	191.000000	192.000000	192.000000
max	255.000000	255.000000	255.000000

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5052 entries, 0 to 5051
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   red     5052 non-null   int64
 1   green   5052 non-null   int64
 2   blue    5052 non-null   int64
 3   label   5052 non-null   object
dtypes: int64(3), object(1)
memory usage: 158.0+ KB
```

Pre Process, Clean the Data

```
#To find whether null entries are present
dataset.isnull().sum()
```

```
red      0
green    0
blue     0
label    0
dtype: int64
```

```
dataset.label.unique()
```

```
array(['Blue', 'Brown', 'Green', 'Pink', 'Yellow', 'Orange', 'Purple',
       'Red', 'Grey', 'White', 'Black'], dtype=object)
```

```
# One-Hot-Encoding
dataset = pd.get_dummies(dataset, columns=['label'])
dataset
```

	red	green	blue	label_Black	label_Blue	label_Brown	label_Green	label_Grey	]
0	20	139	240	False	True	False	False	False	
1	174	83	72	False	False	True	False	False	
2	144	249	131	False	False	False	True	False	
3	168	25	156	False	False	False	False	False	
4	30	182	136	False	False	False	True	False	
...	...	...	...	...	...	...	...	...	
5047	26	26	26	True	False	False	False	False	
5048	27	27	27	True	False	False	False	False	
5049	28	28	28	True	False	False	False	False	
5050	29	29	29	True	False	False	False	False	
5051	30	30	30	True	False	False	False	False	

5052 rows × 14 columns

Next steps: Generate code with dataset View recommended plots

Reorder

```
dataset = dataset[['red', 'green', 'blue', 'label_Red', 'label_Green', 'label_Blue', 'label_Yellow', 'label_Orange', 'label_Pink', 'label_P
dataset
```

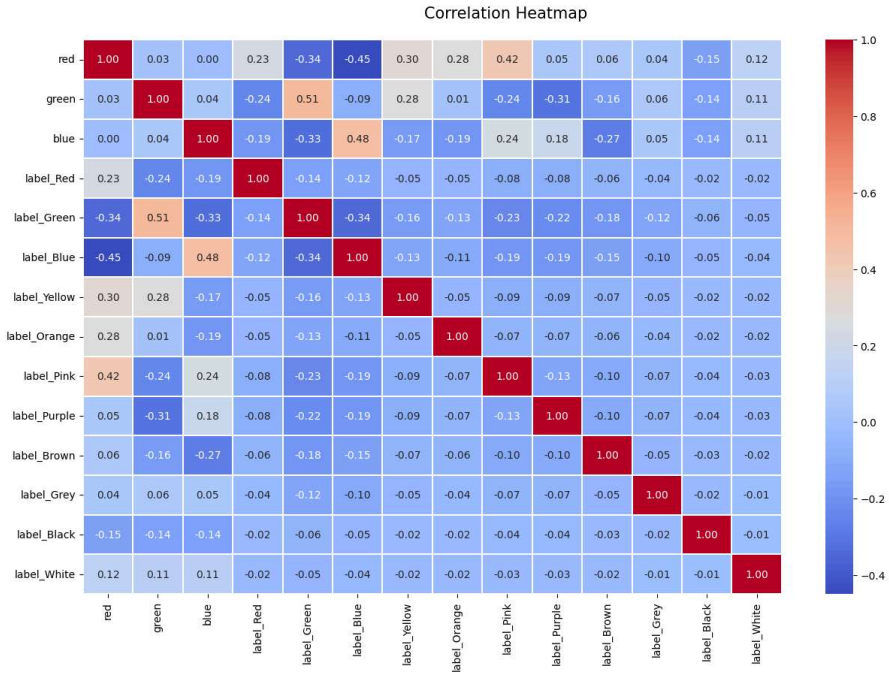
	red	green	blue	label_Red	label_Green	label_Blue	label_Yellow	label_Orange
0	20	139	240	False	False	True	False	False
1	174	83	72	False	False	False	False	False
2	144	249	131	False	True	False	False	False
3	168	25	156	False	False	False	False	False
4	30	182	136	False	True	False	False	False
...	...	...	...	...	...	...	...	...
5047	26	26	26	False	False	False	False	False
5048	27	27	27	False	False	False	False	False
5049	28	28	28	False	False	False	False	False
5050	29	29	29	False	False	False	False	False
5051	30	30	30	False	False	False	False	False

5052 rows × 14 columns

Next steps: Generate code with dataset View recommended plots

Data Correlation

```
# Correlation Matrix Heatmap
f, ax = plt.subplots(figsize=(15, 9))
corr = dataset.corr()
hm = sns.heatmap(round(corr,2), annot=True, ax=ax, cmap="coolwarm",fmt='.2f',
                 linewidths=.05)
f.subplots_adjust(top=0.93)
t= f.suptitle('Correlation Heatmap', fontsize=15)
```



Split the data into train and test

```
train_dataset = dataset.sample(frac=0.8, random_state=9) #train = 80%,  random_state = any int value means every time when you run your pro
test_dataset = dataset.drop(train_dataset.index) #remove train_dataset from dataframe to get test_dataset
train_dataset
```

	red	green	blue	label_Red	label_Green	label_Blue	label_Yellow	label_Orange
4232	243	2	187	False	False	False	False	False
4898	200	89	0	False	False	False	False	True
1959	147	118	80	False	False	False	False	False
3477	30	103	73	False	True	False	False	False
4167	222	205	78	False	False	False	True	False
...	...	...	...	...	...	...	...	...
2414	42	8	85	False	False	True	False	False
2610	162	159	216	False	False	True	False	False
1739	138	48	233	False	False	False	False	False
3033	9	46	248	False	False	True	False	False
1601	141	244	244	False	False	True	False	False

4042 rows × 14 columns

Next steps: Generate code with `train_dataset` View recommended plots

```
test_dataset
```

	red	green	blue	label_Red	label_Green	label_Blue	label_Yellow	label_Orange
0	20	139	240	False	False	True	False	False
1	174	83	72	False	False	False	False	False
3	168	25	156	False	False	False	False	False
11	249	99	108	False	False	False	False	False
12	24	161	177	False	False	True	False	False
...	...	...	...	...	...	...	...	...
5039	18	18	18	False	False	False	False	False
5040	19	19	19	False	False	False	False	False
5042	21	21	21	False	False	False	False	False
5048	27	27	27	False	False	False	False	False
5049	28	28	28	False	False	False	False	False

1010 rows × 14 columns

Next steps: Generate code with `test_dataset` View recommended plots

Split features: red, green, blue and labels

```
train_labels = pd.DataFrame([train_dataset.pop(x) for x in ['label_Red', 'label_Green', 'label_Blue', 'label_Yellow', 'label_Orange', 'labe
train_labels
```

	label_Red	label_Green	label_Blue	label_Yellow	label_Orange	label_Pink	label_
4232	False	False	False	False	False	True	
4898	False	False	False	False	True	False	
1959	False	False	False	False	False	False	
3477	False	True	False	False	False	False	
4167	False	False	False	True	False	False	
...	...	...	...	...	...	...	
2414	False	False	True	False	False	False	
2610	False	False	True	False	False	False	
1739	False	False	False	False	False	False	
3033	False	False	True	False	False	False	
1601	False	False	True	False	False	False	

4042 rows × 11 columns

Next steps: Generate code with `train_labels` View recommended plots

```
train_dataset
```

	red	green	blue
4232	243	2	187
4898	200	89	0
1959	147	118	80
3477	30	103	73
4167	222	205	78
...	...	...	...
2414	42	8	85
2610	162	159	216
1739	138	48	233
3033	9	46	248
1601	141	244	244

4042 rows × 3 columns

Next steps: Generate code with `train_dataset` View recommended plots

```
test_labels = pd.DataFrame([test_dataset.pop(x) for x in ['label_Red', 'label_Green', 'label_Blue', 'label_Yellow', 'label_Orange', 'label_
test_labels
```

	label_Red	label_Green	label_Blue	label_Yellow	label_Orange	label_Pink	label_
0	False	False	True	False	False	False	
1	False	False	False	False	False	False	
3	False	False	False	False	False	True	
11	False	False	False	False	False	True	
12	False	False	True	False	False	False	
...	...	...	...	...	...	...	
5039	False	False	False	False	False	False	
5040	False	False	False	False	False	False	
5042	False	False	False	False	False	False	
5048	False	False	False	False	False	False	
5049	False	False	False	False	False	False	

1010 rows × 11 columns

Next steps: Generate code with `test_labels` View recommended plots

```
test_dataset
```

	red	green	blue
0	20	139	240
1	174	83	72
3	168	25	156
11	249	99	108
12	24	161	177
...	...	...	...
5039	18	18	18
5040	19	19	19
5042	21	21	21
5048	27	27	27
5049	28	28	28

1010 rows × 3 columns

Next steps: Generate code with `test_dataset` View recommended plots

Build & Compile the Model

```
from keras import regularizers
model = keras.Sequential([
    layers.Dense(3, kernel_regularizer=regularizers.l2(0.001), activation='relu', input_shape=[len(train_dataset.keys())]), #inputshape=[3]
    layers.Dense(24, kernel_regularizer=regularizers.l2(0.001), activation='relu'),
    layers.Dense(11)
  ])
```

Compile the model

```
optimizer = keras.optimizers.Adam(learning_rate=0.001)
loss_function = tf.keras.losses.CategoricalCrossentropy(from_logits=True)

model.compile(loss=loss_function,
                optimizer=optimizer,
                metrics=['accuracy'])
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 3)                 12

 dense_1 (Dense)             (None, 24)                96

 dense_2 (Dense)             (None, 11)                275

=================================================================
Total params: 383 (1.50 KB)
Trainable params: 383 (1.50 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Train the model

```
history = model.fit(x=train_dataset, y=train_labels,
                    validation_split=0.3,
                    epochs=10001,
                    batch_size=48,
                    verbose=0,
                    callbacks=[tfdocs.modeling.EpochDots()],
                    shuffle=True)
```

```
Epoch: 7300, accuracy:0.8257,  loss:0.4254,  val_accuracy:0.8021,  val_loss:0.5620,
.....................................................................................
Epoch: 7400, accuracy:0.8300,  loss:0.4243,  val_accuracy:0.7881,  val_loss:0.5713,
.....................................................................................
Epoch: 7500, accuracy:0.8321,  loss:0.4080,  val_accuracy:0.7955,  val_loss:0.6012,
.....................................................................................
Epoch: 7600, accuracy:0.8271,  loss:0.4209,  val_accuracy:0.7955,  val_loss:0.5623,
.....................................................................................
Epoch: 7700, accuracy:0.8321,  loss:0.4081,  val_accuracy:0.8021,  val_loss:0.5643,
.....................................................................................
Epoch: 7800, accuracy:0.8310,  loss:0.4102,  val_accuracy:0.7939,  val_loss:0.5937,
.....................................................................................
Epoch: 7900, accuracy:0.8286,  loss:0.4098,  val_accuracy:0.7955,  val_loss:0.5833,
.....................................................................................
Epoch: 8000, accuracy:0.8374,  loss:0.4055,  val_accuracy:0.7980,  val_loss:0.5932,
.....................................................................................
Epoch: 8100, accuracy:0.8303,  loss:0.4087,  val_accuracy:0.7988,  val_loss:0.5852,
.....................................................................................
Epoch: 8200, accuracy:0.8321,  loss:0.4080,  val_accuracy:0.8005,  val_loss:0.5960,
.....................................................................................
Epoch: 8300, accuracy:0.8356,  loss:0.4071,  val_accuracy:0.7865,  val_loss:0.5715,
.....................................................................................
Epoch: 8400, accuracy:0.8314,  loss:0.4098,  val_accuracy:0.8046,  val_loss:0.5810,
.....................................................................................
Epoch: 8500, accuracy:0.8335,  loss:0.4098,  val_accuracy:0.7931,  val_loss:0.6015,
.....................................................................................
Epoch: 8600, accuracy:0.8332,  loss:0.4130,  val_accuracy:0.8005,  val_loss:0.5757,
.....................................................................................
Epoch: 8700, accuracy:0.8367,  loss:0.4112,  val_accuracy:0.7939,  val_loss:0.5753,
.....................................................................................
Epoch: 8800, accuracy:0.8264,  loss:0.4079,  val_accuracy:0.8013,  val_loss:0.5719,
.....................................................................................
Epoch: 8900, accuracy:0.8367,  loss:0.4122,  val_accuracy:0.7972,  val_loss:0.5705,
.....................................................................................
Epoch: 9000, accuracy:0.8342,  loss:0.4088,  val_accuracy:0.7988,  val_loss:0.5950,
.....................................................................................
Epoch: 9100, accuracy:0.8317,  loss:0.4081,  val_accuracy:0.7980,  val_loss:0.5897,
.....................................................................................
Epoch: 9200, accuracy:0.8307,  loss:0.4093,  val_accuracy:0.7964,  val_loss:0.5824,
.....................................................................................
Epoch: 9300, accuracy:0.8342,  loss:0.4084,  val_accuracy:0.7988,  val_loss:0.5737,
.....................................................................................
Epoch: 9400, accuracy:0.8360,  loss:0.4121,  val_accuracy:0.7997,  val_loss:0.5950,
.....................................................................................
Epoch: 9500, accuracy:0.8317,  loss:0.4230,  val_accuracy:0.8054,  val_loss:0.5820,
.....................................................................................
Epoch: 9600, accuracy:0.8346,  loss:0.4117,  val_accuracy:0.7890,  val_loss:0.5788,
.....................................................................................
Epoch: 9700, accuracy:0.8346,  loss:0.4093,  val_accuracy:0.7931,  val_loss:0.5835,
.....................................................................................
Epoch: 9800, accuracy:0.8317,  loss:0.4109,  val_accuracy:0.7890,  val_loss:0.5901,
.....................................................................................
Epoch: 9900, accuracy:0.8310,  loss:0.4100,  val_accuracy:0.8071,  val_loss:0.5800,
.....................................................................................
Epoch: 10000, accuracy:0.8314,  loss:0.4113,  val_accuracy:0.8038,  val_loss:0.5816,
.
```

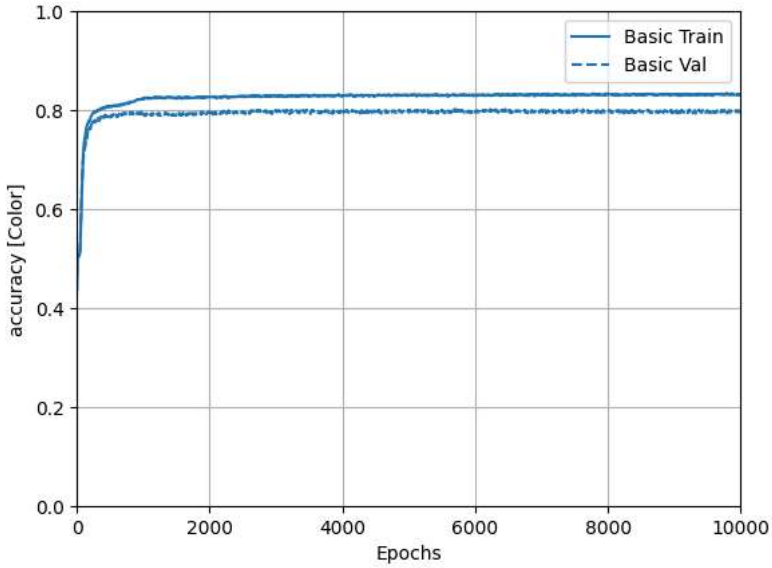
```
hist = pd.DataFrame(history.history)
hist['epoch'] = history.epoch
hist.tail()
```

	loss	accuracy	val_loss	val_accuracy	epoch
9996	0.405345	0.834924	0.598833	0.802143	9996
9997	0.411392	0.829975	0.569255	0.798846	9997
9998	0.415667	0.827501	0.627256	0.783182	9998
9999	0.419921	0.830329	0.571426	0.795548	9999
10000	0.411254	0.831389	0.581584	0.803792	10000

```
plotter = tfdocs.plots.HistoryPlotter(smoothing_std=2)
```

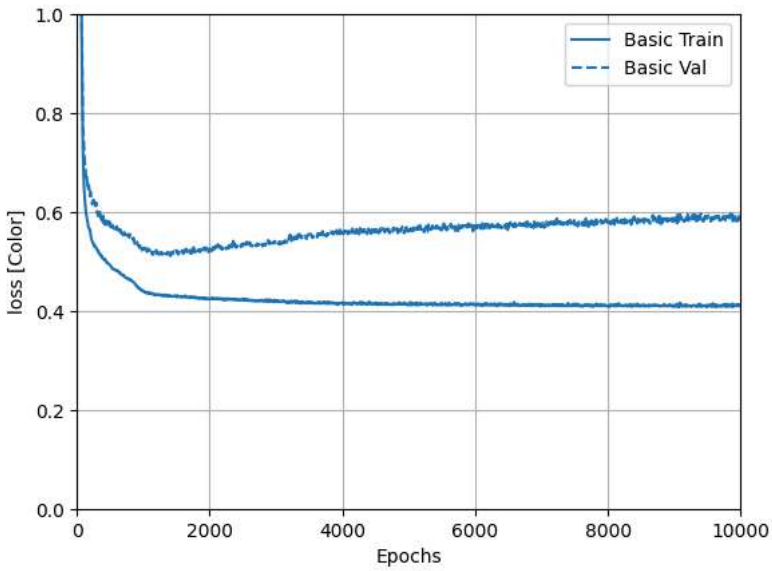
```
plotter.plot({'Basic': history}, metric = "accuracy")
plt.ylim([0, 1])
plt.ylabel('accuracy [Color]')
```

```
Text(0, 0.5, 'accuracy [Color]')
```




```
plotter.plot({'Basic': history}, metric = "loss")
plt.ylim([0, 1])
plt.ylabel('loss [Color]')
```

```
Text(0, 0.5, 'loss [Color]')
```



Save Model

```
#Install PyDrive library into Google Colab notebook
!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
```

```
WARNING:root:pydrive is deprecated and no longer maintained. We recommend that you migrate your projects to pydrive2, the maintained fo
```

```
#Authenticate and create the PyDrive client
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

```
!pip install pyyaml h5py  # Required to save models in HDF5 format
```

```
# Save Model or weights on google drive and create on Colab directory in Google Drive

# Save the entire model to a HDF5 file.
# The '.h5' extension indicates that the model should be saved to HDF5.
model.save('colormodel_acu_89.h5')
model_file = drive.CreateFile({'title' : 'colormodel_acu_89.h5'}) # colormodel_acu_  .h5 is the name of the model uploaded to Drive
model_file.SetContentFile('colormodel_acu_89.h5')
model_file.Upload()
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning:

You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the nat
```

```
# download to google drive
drive.CreateFile({'id': model_file.get('id')}) ### very important to run this last cell for loading the model in new notebook
```

```
GoogleDriveFile({'id': '1b2xTIZ6M5W8_CuAEftRaTaufPdzkkFlE'})
```

```
#Load model from google drive into colab
file_obj = drive.CreateFile({'id': '1b2xTIZ6M5W8_CuAEftRaTaufPdzkkFlE'})
file_obj.GetContentFile('colormodel_trained_89.h5') #colormodel_trained.h5 is the name of model uploaded to colab
```

```
ls
```

```
colormodel_acu_89.h5  colormodel_trained_89.h5  final_data.csv  sample_data/
```

```
# Recreate the exact same model, including its weights and the optimizer
model = tf.keras.models.load_model('colormodel_trained_89.h5') #very important
```

```
# Show the model architecture
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 3)                 12

 dense_1 (Dense)             (None, 24)                96

 dense_2 (Dense)             (None, 11)                275

=================================================================
Total params: 383 (1.50 KB)
Trainable params: 383 (1.50 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Make Prediction

The prediction by the ANN model is an array of 11 numbers.

They represent the model's "confidence" that the RGB color corresponds to each of the 11 different classes of color as follows:

*0 for Red

*1 for Green

*2 for Blue

*3 for Yellow

*4 for Orange

*5 for Pink

*6 for Purple

*7 for Brown

*8 for Grey

*9 for Black

*10 for White

You can see which out of the 11 labels has the highest confidence value

Train Dataset Prediction

```
train_predictions = model.predict(train_dataset)
train_predictions
```

```
127/127 [==============================] - 1s 3ms/step
array([[-3.0546806e+00, -1.3563481e+02, -2.5335173e+01, ...,
        -2.9971249e+01, -9.3503888e+02, -3.4184658e+01],
       [-8.7259769e-02, -2.3787195e+01, -2.2071377e+01, ...,
        -1.2348607e+01, -7.9312207e+02, -1.3869256e+01],
       [-1.1208030e+02, -8.4821701e-02, -7.1351185e+00, ...,
         7.1456730e-01, -2.4825415e+02, -4.7922087e+00],
       ...,
       [-1.3734222e+01, -2.8312176e+01,  8.3375269e-01, ...,
        -2.6766834e+00, -7.8002419e+01, -2.1341066e+01],
       [-1.6776926e+02, -8.7298179e-01,  8.8513756e+00, ...,
        -1.8164547e+01, -5.5702221e+01, -3.1905197e+01],
       [-2.6176837e+02,  4.4423938e+00,  5.9071903e+00, ...,
        -2.4778128e-01, -2.8633469e+01, -3.3689114e+01]], dtype=float32)
```

Selecting Class with highest confidence

```
actual_encoded_train_labels = np.argmax(train_labels.to_numpy(), axis=1) #train_labels were originally in one-hot
actual_encoded_train_labels
```

```
array([5, 4, 7, ..., 6, 2, 2])
```

```
predicted_encoded_train_labels = np.argmax(train_predictions, axis=1)
predicted_encoded_train_labels
```

```
array([5, 4, 7, ..., 6, 2, 2])
```

Converting numpy array to pandas dataframe

```
actual_encoded_train_labels = pd.DataFrame(actual_encoded_train_labels, columns=['Labels'])
actual_encoded_train_labels
```

	Labels
0	5
1	4
2	7
3	1
4	3
...	...
4037	2
4038	2
4039	6
4040	2
4041	2

4042 rows × 1 columns

Next steps: Generate code with `actual_encoded_train_labels` View recommended plots

```
predicted_encoded_train_labels = pd.DataFrame(predicted_encoded_train_labels, columns=['Labels'])
predicted_encoded_train_labels
```

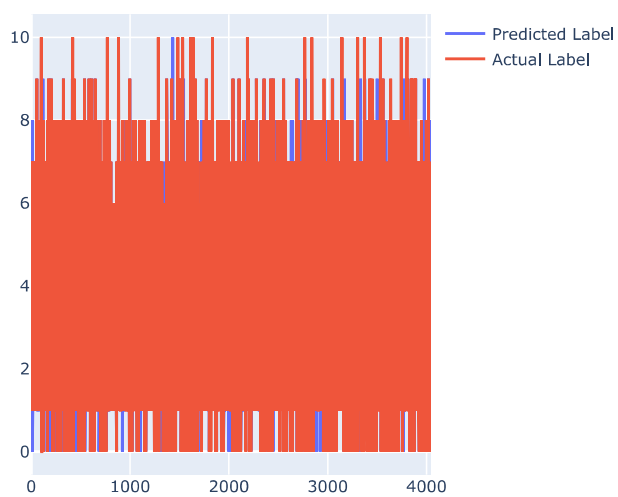
	Labels
0	5
1	4
2	7
3	1
4	3
...	...
4037	6
4038	6
4039	6
4040	2
4041	2

4042 rows × 1 columns

Next steps: Generate code with `predicted_encoded_train_labels` View recommended plots

Visualize Prediction for Train Dataset

```
#Plot Actual vs Predicted Class for Training Dataset
actual_chart = go.Scatter(x=actual_encoded_train_labels.index, y=actual_encoded_train_labels.Labels, name= 'Actual Label')
predict_chart = go.Scatter(x=actual_encoded_train_labels.index, y=predicted_encoded_train_labels.Labels, name= 'Predicted Label')
py.iplot([predict_chart, actual_chart])
```



Test Dataset Prediction

```
test_predictions = model.predict(test_dataset)
test_predictions
```

```
32/32 [==============================] - 0s 2ms/step
array([[-1.6776926e+02, -8.7298179e-01,  8.8513756e+00, ...,
        -1.8164547e+01, -5.5702221e+01, -3.1905197e+01],
       [-5.6559849e-01, -3.7443626e+01, -1.6782183e+01, ...,
        -8.4941454e+00, -5.1382312e+02, -8.3214512e+00],
       [-8.2696218e+00, -7.9242432e+01, -1.1817730e+01, ...,
        -1.5804877e+01, -4.8399951e+02, -2.7377563e+01],
       ...,
       [-2.4205118e+02,  3.0333428e+00,  2.3390055e-03, ...,
```

```
  4.9305677e+00,  7.9414210e+00, -2.6698792e+01],
[-2.3402168e+02,  2.9281507e+00, -3.1423390e-02, ...,
  5.0796547e+00,  7.0221953e+00, -2.1306261e+01],
[-2.3268344e+02,  2.9106185e+00, -3.7048757e-02, ...,
  5.1045027e+00,  6.8689885e+00, -2.0407482e+01]], dtype=float32)
```

Selecting Class with highest confidence

```
actual_encoded_test_labels = np.argmax(test_labels.to_numpy(), axis=1)
actual_encoded_test_labels
```

```
array([2, 7, 5, ..., 9, 9, 9])
```

```
predicted_encoded_test_labels = np.argmax(test_predictions, axis=1)
predicted_encoded_test_labels
```

```
array([2, 7, 5, ..., 9, 9, 9])
```

Converting numpy array to pandas dataframe

```
actual_encoded_test_labels = pd.DataFrame(actual_encoded_test_labels, columns=['Labels'])
actual_encoded_test_labels
```

	Labels
0	2
1	7
2	5
3	5
4	2
...	...
1005	9
1006	9
1007	9
1008	9
1009	9

1010 rows × 1 columns

Next steps: Generate code with `actual_encoded_test_labels` View recommended plots

```
predicted_encoded_test_labels = pd.DataFrame(predicted_encoded_test_labels, columns=['Labels'])
predicted_encoded_test_labels
```

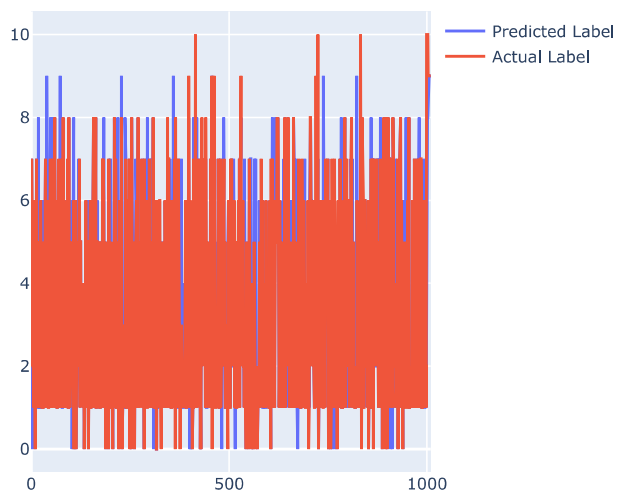
	Labels
0	2
1	7
2	5
3	0
4	2
...	...
1005	9
1006	9
1007	9
1008	9
1009	9

1010 rows × 1 columns

Next steps: Generate code with predicted_encoded_test_labels View recommended plots

Visualize Prediction for Test Dataset

```
#Plot Actual vs Predicted Class for Test Dataset
actual_chart = go.Scatter(x=actual_encoded_test_labels.index, y=actual_encoded_test_labels.Labels, name= 'Actual Label')
predict_chart = go.Scatter(x=actual_encoded_test_labels.index, y=predicted_encoded_test_labels.Labels, name= 'Predicted Label')
py.iplot([predict_chart, actual_chart])
```



Evaluate Model

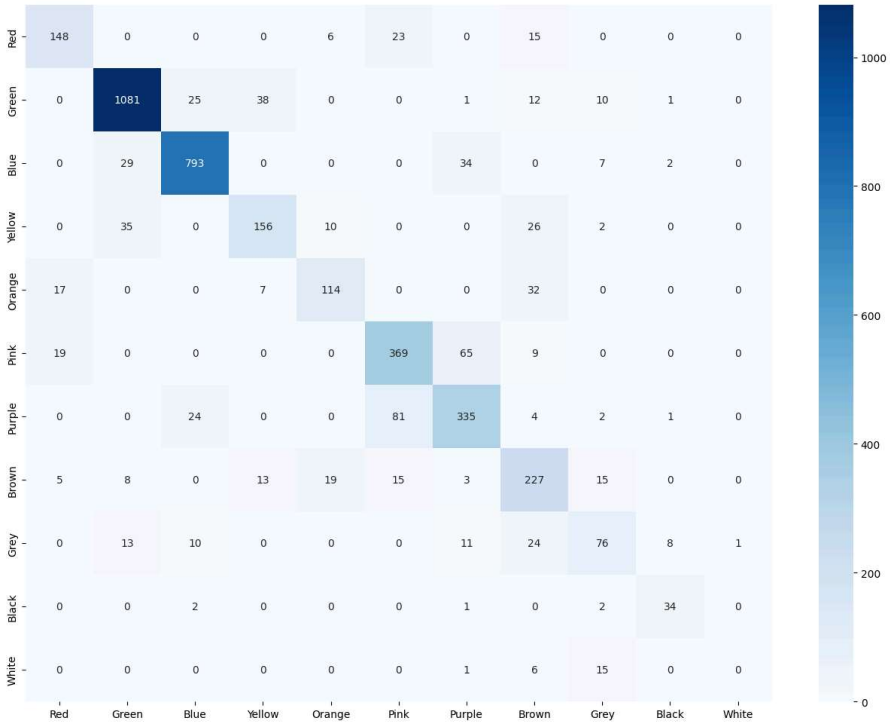
```
model.evaluate(x=train_dataset, y=train_labels)
```

```
127/127 [==============================] - 1s 7ms/step - loss: 0.4549 - accuracy: 0.8246
[0.45487523078918457, 0.8245918154716492]
```

```
from sklearn.metrics import confusion_matrix, classification_report
confusion_matrix_train = confusion_matrix(actual_encoded_train_labels, predicted_encoded_train_labels)
confusion_matrix_train
```

```
array([[ 148,    0,    0,    0,    6,   23,    0,   15,    0,    0,    0],
       [   0, 1081,   25,   38,    0,    0,    1,   12,   10,    1,    0],
       [   0,   29,  793,    0,    0,    0,   34,    0,    7,    2,    0],
       [   0,   35,    0,  156,   10,    0,    0,   26,    2,    0,    0],
       [  17,    0,    0,    7,  114,    0,    0,   32,    0,    0,    0],
       [  19,    0,    0,    0,    0,  369,   65,    9,    0,    0,    0],
       [   0,    0,   24,    0,    0,   81,  335,    4,    2,    1,    0],
       [   5,    8,    0,   13,   19,   15,    3,  227,   15,    0,    0],
       [   0,   13,   10,    0,    0,    0,   11,   24,   76,    8,    1],
       [   0,    0,    2,    0,    0,    0,    1,    0,    2,   34,    0],
       [   0,    0,    0,    0,    0,    0,    1,    6,   15,    0,    0]])
```

```
f,ax = plt.subplots(figsize=(16,12))
categories = ['Red', 'Green', 'Blue', 'Yellow', 'Orange', 'Pink', 'Purple', 'Brown', 'Grey', 'Black', 'White']
sns.heatmap(confusion_matrix_train, annot=True, cmap='Blues', fmt='d',
            xticklabels = categories,
            yticklabels = categories)
plt.show()
```




```
target_names = ['Red', 'Green', 'Blue', 'Yellow', 'Orange', 'Pink', 'Purple', 'Brown', 'Grey', 'Black', 'White']
print(classification_report(actual_encoded_train_labels, predicted_encoded_train_labels, target_names=target_names))
```

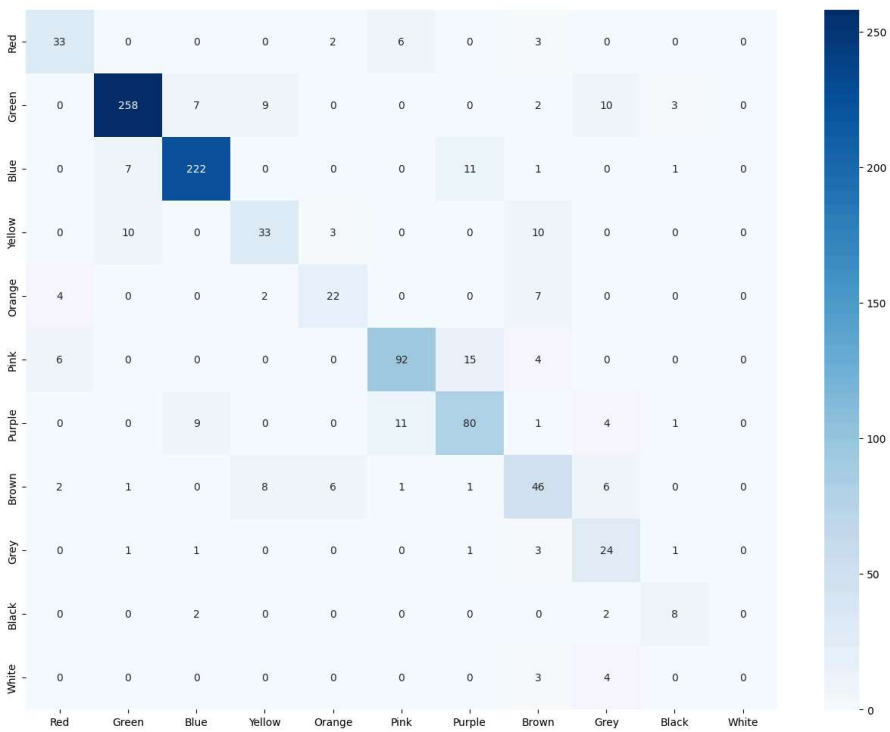
```
              precision    recall  f1-score   support

         Red       0.78      0.77      0.78       192
       Green       0.93      0.93      0.93      1168
        Blue       0.93      0.92      0.92       865
      Yellow       0.73      0.68      0.70       229
      Orange       0.77      0.67      0.71       170
        Pink       0.76      0.80      0.78       462
      Purple       0.74      0.75      0.75       447
       Brown       0.64      0.74      0.69       305
        Grey       0.59      0.53      0.56       143
       Black       0.74      0.87      0.80        39
       White       0.00      0.00      0.00        22

    accuracy                           0.82      4042
   macro avg       0.69      0.70      0.69      4042
weighted avg       0.82      0.82      0.82      4042
```

Evaluating for Test Dataset

```
model.evaluate(x=test_dataset, y=test_labels)
```

```
32/32 [==============================] - 0s 2ms/step - loss: 0.5807 - accuracy: 0.8099
[0.5807260870933533, 0.8099009990692139]
```

```
confusion_matrix_test = confusion_matrix(actual_encoded_test_labels, predicted_encoded_test_labels)
confusion_matrix_test
```

```
array([[ 33,   0,   0,   0,   2,   6,   0,   3,   0,   0,   0],
       [  0, 258,   7,   9,   0,   0,   0,   2,  10,   3,   0],
       [  0,   7, 222,   0,   0,   0,  11,   1,   0,   1,   0],
       [  0,  10,   0,  33,   3,   0,   0,  10,   0,   0,   0],
       [  4,   0,   0,   2,  22,   0,   0,   7,   0,   0,   0],
       [  6,   0,   0,   0,   0,  92,  15,   4,   0,   0,   0],
       [  0,   0,   9,   0,   0,  11,  80,   1,   4,   1,   0],
       [  2,   1,   0,   8,   6,   1,   1,  46,   6,   0,   0],
       [  0,   1,   1,   0,   0,   0,   1,   3,  24,   1,   0],
       [  0,   0,   2,   0,   0,   0,   0,   0,   2,   8,   0],
       [  0,   0,   0,   0,   0,   0,   0,   3,   4,   0,   0]])
```

```
f,ax = plt.subplots(figsize=(16,12))
categories = ['Red', 'Green', 'Blue', 'Yellow', 'Orange', 'Pink', 'Purple', 'Brown', 'Grey', 'Black', 'White']
sns.heatmap(confusion_matrix_test, annot=True, cmap='Blues', fmt='d',
            xticklabels = categories,
            yticklabels = categories)
plt.show()
```