Item Classification using Deep Learning

```
# Mount to Google Drive for downloading dataset file

from google.colab import drive
drive.mount('/content/gdrive/')
```

```
Mounted at /content/gdrive/
```

```
# Unzip the dataset file

!unzip /content/Dataset.zip
```

```
import pandas as pd
import numpy as np
import glob
import os
from datetime import datetime
from packaging import version

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.applications import VGG19
from tensorflow.keras.preprocessing import image_dataset_from_directory
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.callbacks import ModelCheckpoint, History

from keras.models import Sequential, load_model
from keras.layers import Conv2D, Lambda, MaxPooling2D, Dense, Dropout, Flatten
from keras.preprocessing.image import ImageDataGenerator
from keras.utils.np_utils import to_categorical

from skimage.io import imread, imshow
from skimage.transform import resize
from IPython import display
import matplotlib.pyplot as plt
import seaborn as sns
from seaborn import heatmap
from sklearn.metrics import confusion_matrix
```

```
# Define train and test image folder path

train_folder = "/content/Dataset/Train"
test_folder = "/content/Dataset/Test"
```

```
# Print total number of training and testing images

print("Number of training images: {}".format(len(list(glob.glob(os.path.join(trai
print("Number of testing images: {}".format(len(list(glob.glob(os.path.join(test_
```

```
Number of training images: 6146
Number of testing images: 1064
```

```
# Create variables of batch size and image size

batch_size = 64
img_height = 180
img_width = 180
```

```
# Datagenerators
# Apply a rescale parameter in ImageDataGenerator for a rescaling factor to the p
# It is to rescale the pixel values to a specific range in order to normalize the
# rescale = 1.0/255, the pixel values will be divided by 255, resulting in normal
# validation_split = 0.2 meaning 80% of the images are for training and 20% are f

train_datagen = ImageDataGenerator(rescale = 1.0 / 255.0, validation_split = 0.2)

valid_datagen = ImageDataGenerator(rescale = 1.0 / 255.0, validation_split = 0.2)

test_datagen  = ImageDataGenerator(rescale = 1.0 / 255.0)
```

```
# Train dataset

train_ds  = train_datagen.flow_from_directory(directory = train_folder,
                                              target_size = (img_height, img
                                              class_mode = 'categorical',
                                              batch_size = batch_size,
                                              subset = 'training')
```

```
Found 4919 images belonging to 2 classes.
```

```
# Validate dataset

valid_ds = valid_datagen.flow_from_directory(directory = train_folder,
                                             target_size = (img_height, img_
                                             class_mode = 'categorical',
                                             batch_size = batch_size,
                                             subset = 'validation')
```

```
Found 1229 images belonging to 2 classes.
```

```
# Test dataset

test_ds = test_datagen.flow_from_directory(directory = test_folder,
                                                    target_size = (img_height, img_
                                                    class_mode = 'categorical',
                                                    batch_size = batch_size,
                                                    shuffle=False)
```

```
Found 1065 images belonging to 2 classes.
```

```
# Print class values

print(train_ds.class_indices)
print(test_ds.class_indices)
```

```
{'Non-recyclable': 0, 'Recyclable': 1}
{'Non-recyclable': 0, 'Recyclable': 1}
```

```
# View train dataset images sample

fig, ax = plt.subplots(nrows = 2, ncols = 5, figsize = (12,6))

for i in range(2):
    for j in range(5):
        rand1 = np.random.randint(len(train_ds))
        rand2 = np.random.randint(batch_size)
        ax[i,j].imshow(train_ds[rand1][0][rand2])
        ax[i,j].axis('off')
        label = train_ds[rand1][1][rand2]
        if label[0] == 0:
            ax[i,j].set_title('Recyclable')
        else:
            ax[i,j].set_title('Non-recyclable')

plt.tight_layout
plt.show()
```


```
# Define callback
# Monitor the accuracy of validation and save the model's weights during training

model_weight_filepath = '/content/assignment_2_model_weights.hdf5'

checkpoint = ModelCheckpoint(model_weight_filepath,
                             monitor = 'val_auc',
                             mode='max',
                             save_best_only=True,
                             verbose = 1)

# Create a TensorBoard callback

!rm -rf ./logs/

root_logdir = "logs"
run_id = datetime.now().strftime("%Y%m%d-%H%M%S")
logdir = os.path.join(root_logdir, run_id)

tboard_callback = tf.keras.callbacks.TensorBoard(log_dir = logdir,
                                                 histogram_freq = 0,
                                                 profile_batch = '500,520')

callback_list = [checkpoint,tboard_callback]
```

```
# Base model - VGG19

base_model = VGG19(input_shape=(180,180,3),
                   include_top=False,
                   weights="imagenet")
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applica
80134624/80134624 [==============================] - 1s 0us/step
```

```
# Set base model layers to not trainable

for layer in base_model.layers:
    layer.trainable=False
```

```
# Model creation

def make_model():

  # Define layers

  model=Sequential()
  model.add(base_model)
  model.add(Dropout(0.5))
  model.add(Flatten())

  # Add dense layers
  #   The he_uniform initializer draws the initial weights from a uniform distrib
  #   where the range of the distribution is determined by the number of input un
  #   It is very effective when used with relu activation functionn
  # Add dropout layers, set to 50% dropout

  model.add(Dense(5000,activation="relu",kernel_initializer='he_uniform'))
  model.add(Dropout(0.5))
  model.add(Dense(1000,activation="relu",kernel_initializer='he_uniform'))
  model.add(Dropout(0.5))
  model.add(Dense(500,activation="relu",kernel_initializer='he_uniform'))
  model.add(Dropout(0.5))
  model.add(Dense(2,activation="softmax"))

  return model

model = make_model()
```

```
# Show model summary with custom layers

model.summary()
```

```
# Model fitting (training)

epochs_num = 30
train_ds_sample_num = train_ds.samples
valid_ds_sample_num = valid_ds.samples

model.compile(
    loss = "categorical_crossentropy",
    optimizer = "adam",
    metrics = [tf.keras.metrics.AUC(name = 'auc')])

model_history = model.fit(
   train_ds,
   epochs = epochs_num,
   steps_per_epoch = int(train_ds_sample_num / batch_size),
   validation_data = valid_ds,
   validation_steps = int(valid_ds_sample_num / batch_size),
   callbacks = callback_list)
```

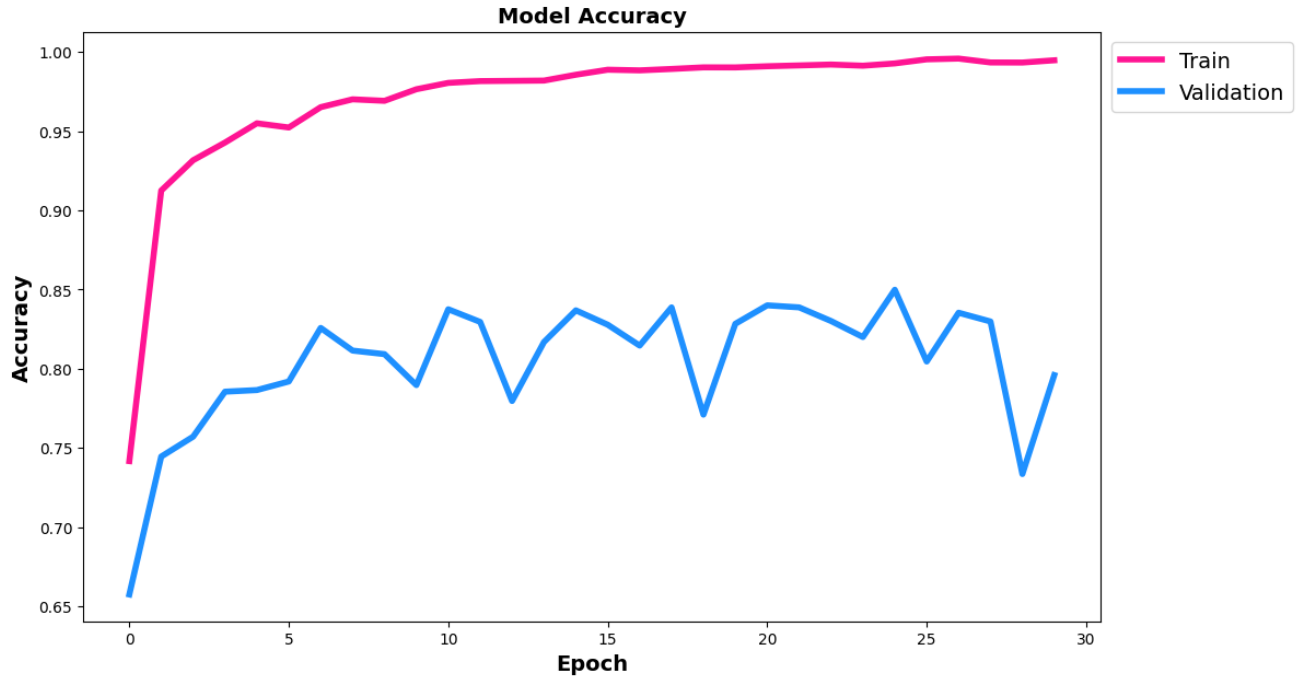
Model Training Execution Time: 16 mins 50 sec (1010 sec)

```
# Store model fitting results in a variable

model_fit_result = model_history

# Save  DataFrame for plotting chart

model_history_df = pd.DataFrame(model_fit_result.history)
model_history_df
```

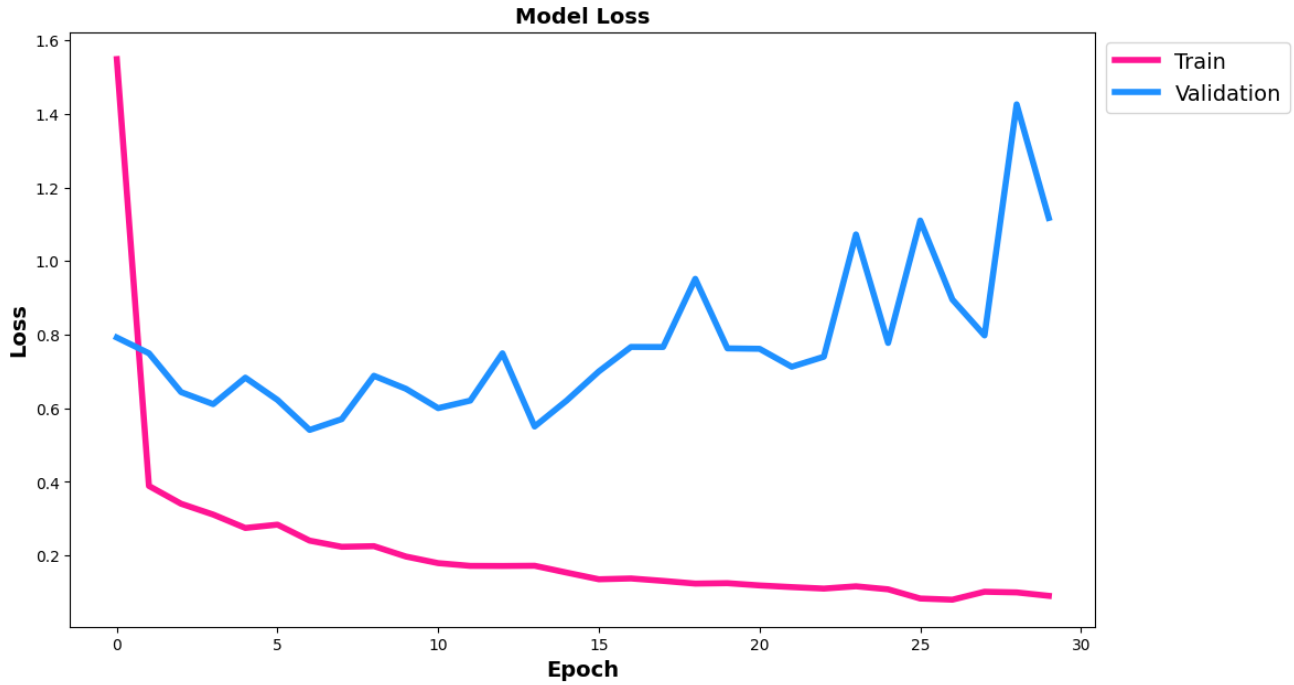
	loss	auc	val_loss	val_auc
0	1.549400	0.741718	0.792426	0.657543
1	0.388895	0.912455	0.749701	0.744579
2	0.340470	0.931594	0.643959	0.757137
3	0.311015	0.942761	0.611272	0.785529
4	0.274603	0.954917	0.683262	0.786545
5	0.283620	0.952262	0.622716	0.791961
6	0.240170	0.965089	0.541153	0.825774
7	0.223387	0.970047	0.570569	0.811462
8	0.225129	0.969095	0.688381	0.809224
9	0.196884	0.976406	0.652868	0.789693
10	0.178979	0.980414	0.600419	0.837536
11	0.171521	0.981537	0.620955	0.829558
12	0.171281	0.981702	0.749500	0.779626
13	0.171854	0.981876	0.550454	0.816854
14	0.153222	0.985550	0.621130	0.836864
15	0.135084	0.988727	0.700311	0.827790
16	0.137292	0.988317	0.766793	0.814622
17	0.130672	0.989205	0.766642	0.838736
18	0.123351	0.990209	0.952089	0.771041
19	0.124472	0.990178	0.762868	0.828367
20	0.118377	0.990915	0.761760	0.840021
21	0.113811	0.991439	0.712925	0.838734
22	0.109785	0.991997	0.740174	0.830125
23	0.115988	0.991256	1.072777	0.820017
24	0.107765	0.992702	0.777644	0.849859
25	0.082801	0.995274	1.110474	0.804557
26	0.079604	0.995758	0.895760	0.835358
27	0.101168	0.993305	0.798003	0.829828
28	0.099348	0.993257	1.426484	0.733545
29	0.090005	0.994677	1.117519	0.795982

Model Performance - Validation Accuracy

```
# Plot the model accuracy

plt.figure(figsize=(12,7))
plt.plot(model_history_df['auc'], color='deeppink', linewidth=4)
plt.plot(model_history_df['val_auc'], color='dodgerblue', linewidth=4)
plt.title('Model Accuracy', fontsize=14, fontweight='bold')
plt.ylabel('Accuracy', fontsize=14, fontweight='bold')
plt.xlabel('Epoch', fontsize=14, fontweight='bold')
plt.legend(['Train', 'Validation'], loc='upper left', bbox_to_anchor=(1,1), fonts
plt.show()
```

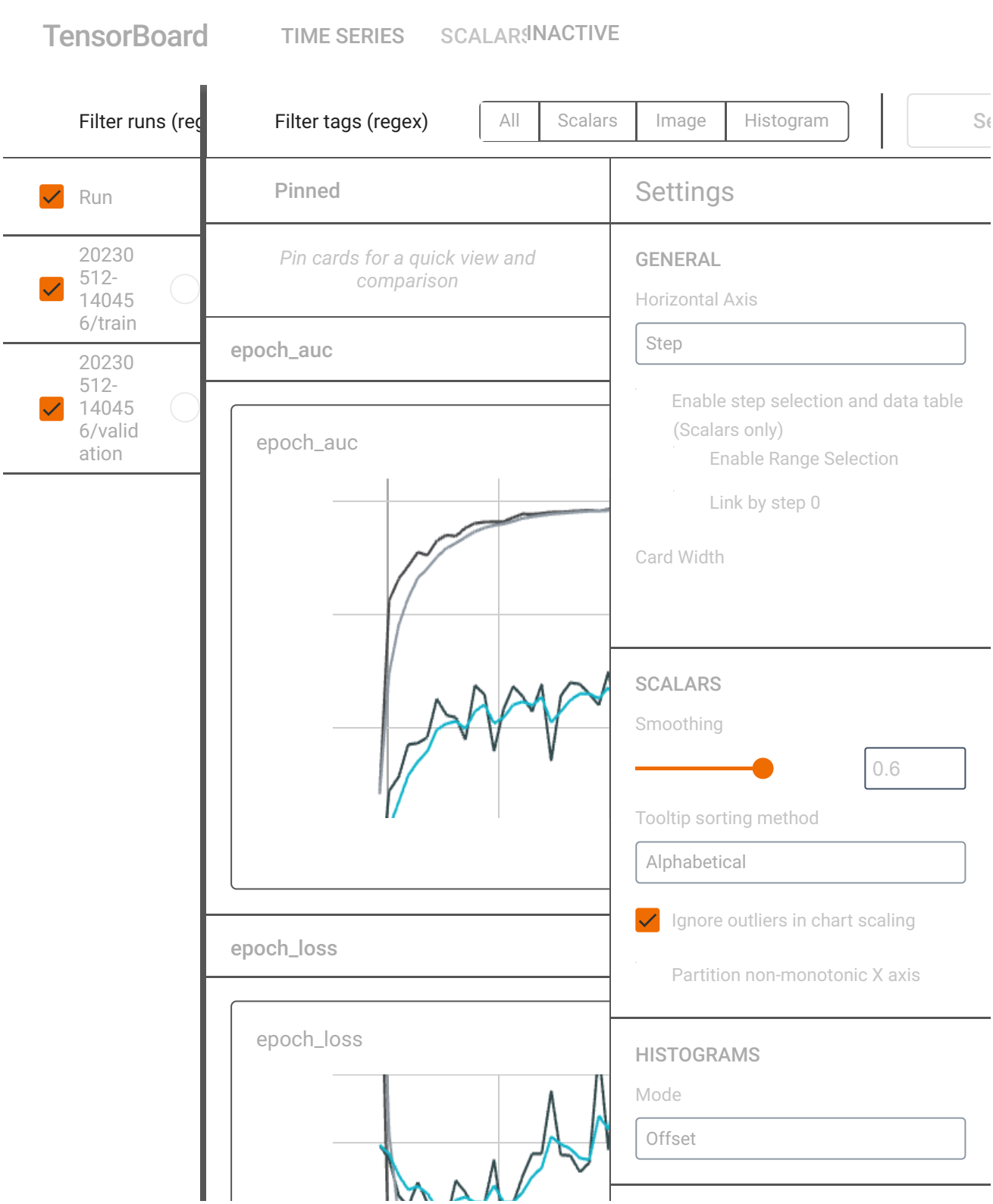



```
# Plot the model loss

plt.figure(figsize=(12,7))
plt.plot(model_history_df['loss'], color='deeppink', linewidth=4)
plt.plot(model_history_df['val_loss'], color='dodgerblue', linewidth=4)
plt.title('Model Loss', fontsize=14, fontweight='bold')
plt.ylabel('Loss', fontsize=14, fontweight='bold')
plt.xlabel('Epoch', fontsize=14, fontweight='bold')
plt.legend(['Train', 'Validation'], loc='upper left', bbox_to_anchor=(1,1), fonts
plt.show()
```




```
# Load the TensorBoard notebook extension.

%reload_ext tensorboard
```

```
# Launch TensorBoard and navigate to the Profile tab to view performance profile

%tensorboard --logdir=logs
```



Model Performance - Evaluation Accuracy

```
# Evaluate overall loss and accuracy for test data

evaluate_loss, evaluate_accuracy = model.evaluate(test_ds)

print(f"The accuracy of model evaluation was {evaluate_accuracy}.")
print(f"The loss of model evaluation was {evaluate_loss}.")
```

```
17/17 [==============================] - 8s 451ms/step - loss: 0.9163 - auc: (
The accuracy of model evaluation was 0.8030783534049988.
The loss of model evaluation was 0.9162694215774536.
```

```
# Calculate confusion matrix

metrics = tf.keras.metrics.AUC(name = 'auc')
test_ds_sample_num = test_ds.samples

prediction_batch_size = 32

Y_pred = model.predict(test_ds, test_ds_sample_num // prediction_batch_size + 1)
y_pred = np.argmax(Y_pred, axis=1)
cm = confusion_matrix(test_ds.classes, y_pred)
print('Confusion Matrix')
print(cm)
```

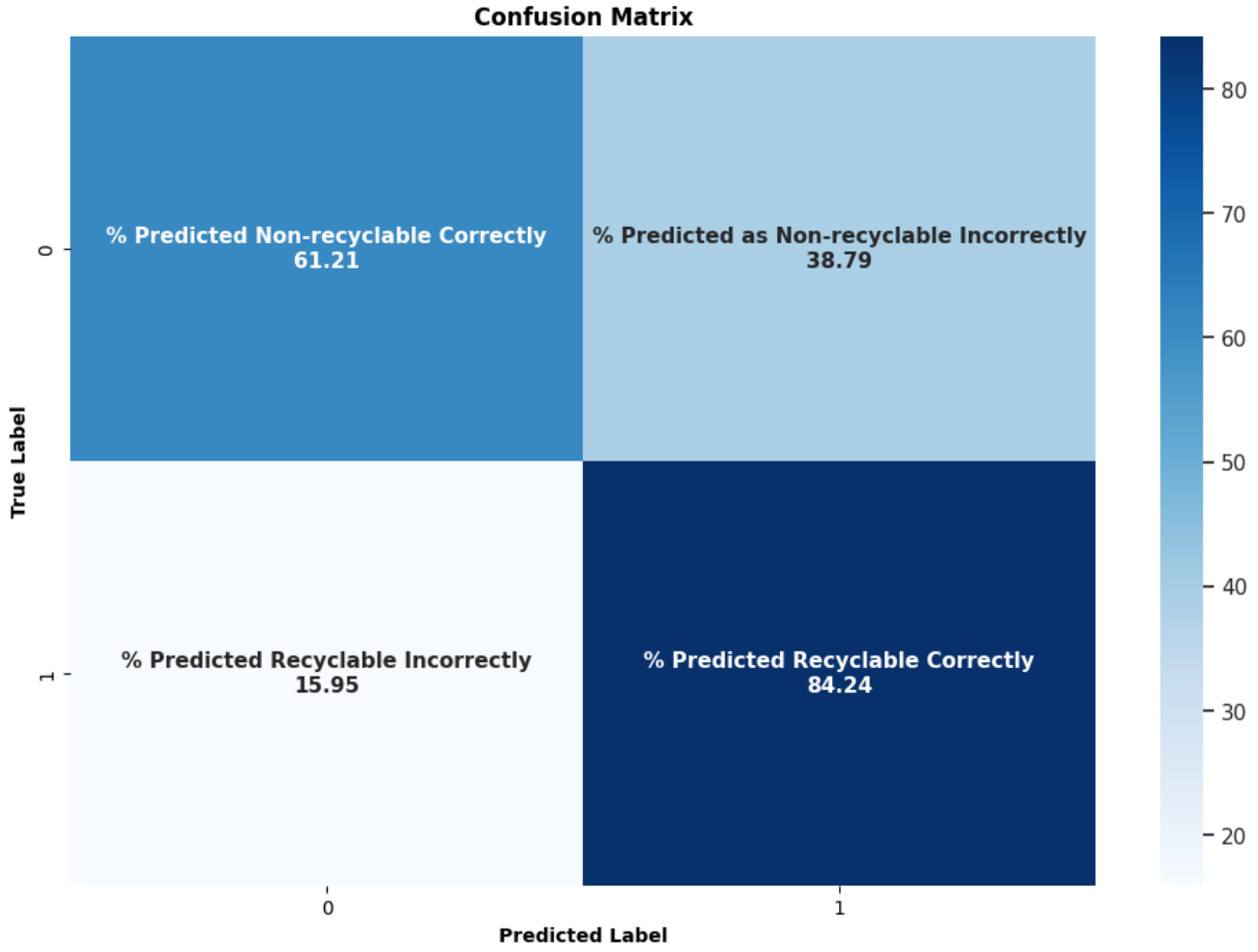
```
17/17 [==============================] - 4s 217ms/step
Confusion Matrix
[[325 206]
 [ 85 449]]
```

```
# Find number of test images in recyclable and non-recyclable class

test_n = glob.glob(os.path.join(test_folder, 'Non-recyclable', '*.jpg'))
test_r = glob.glob(os.path.join(test_folder, 'Recyclable', '*.jpg'))
test_n_num = len(test_n)
test_r_num = len(test_r)

print("Number of test images in non-recyclable class: {}".format(test_n_num))
print("Number of test images in recyclable class: {}".format(test_r_num))
```

```
Number of test images in non-recyclable class: 531
Number of test images in recyclable class: 533
```

```
# Converting confusion matrix to percentage

perc1 = round(cm[0,0] / test_n_num * 100, 2)
perc2 = round(cm[0,1] / test_n_num * 100, 2)
perc3 = round(cm[1,0] / test_r_num * 100, 2)
perc4 = round(cm[1,1] / test_r_num * 100, 2)

# Put calculated numbers into a 2D array

cm_data = np.array([[perc1, perc2],
                [perc3, perc4]])

text = np.array([['% Predicted Non-recyclable Correctly', '% Predicted as Non-rec
                ['% Predicted Recyclable Incorrectly', '% Predicted Recyclable Co

# Combine text with values

formatted_text = (np.asarray(["{0}\n{1:.2f}".format(
text, cm_data) for text, cm_data in zip(text.flatten(), cm_data.flatten())])).res

# Plot heatmap of the confusion matrix

fig, ax = plt.subplots(figsize=(12,8))
sns.set(font_scale=1.0)
ax = sns.heatmap(cm_data, annot=formatted_text, fmt="", cmap='Blues', annot_kws={
ax.set_title("Confusion Matrix", fontsize=12, fontweight='bold')
ax.set_xlabel("Predicted Label", fontsize=10, fontweight='bold')
ax.set_ylabel("True Label", fontsize=10, fontweight='bold')
```

```
Text(120.72222222222221, 0.5, 'True Label')
```



Analyse and improve the model

Build an input pipeline for data augmentation

```
# Apply data augmentation to the train dataset

train_datagen = ImageDataGenerator(rescale = 1.0 / 255.0,
                                   zoom_range = 0.4,
                                   rotation_range = 10,
                                   horizontal_flip = True,
                                   vertical_flip = True,
                                   validation_split = 0.2)

train_ds  = train_datagen.flow_from_directory(directory = train_folder,
                                              target_size = (img_height, img
                                              class_mode = 'categorical',
                                              batch_size = batch_size,
                                              subset = 'training')
```

```
Found 4919 images belonging to 2 classes.
```

Profile the input pipeline to identify the most time-consuming operation

```
!pip install -U tensorboard_plugin_profile
```

```
# Create a TensorBoard callback for TensorFlow Profiler
# logs = "logs/" + datetime.now().strftime("%Y%m%d-%H%M%S")

!rm -rf logs

logs = "logs"

profile_callback = tf.keras.callbacks.TensorBoard(log_dir = logs,
                                                  histogram_freq = 0,
                                                  profile_batch = '500,520')

callback_list = [checkpoint,profile_callback]
```

```
# Re-create model

model = make_model()
```

```
# Re-train model with data augmentation

num_of_epochs = 30
train_ds_sample_num = train_ds.samples
valid_ds_sample_num = valid_ds.samples

model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=[tf.kera

data_aug_model_history = model.fit(
   train_ds,
   epochs=num_of_epochs,
   steps_per_epoch=int(train_ds_sample_num/batch_size),
   validation_data=valid_ds,
   validation_steps=int(valid_ds_sample_num/batch_size),
   callbacks = callback_list)
```

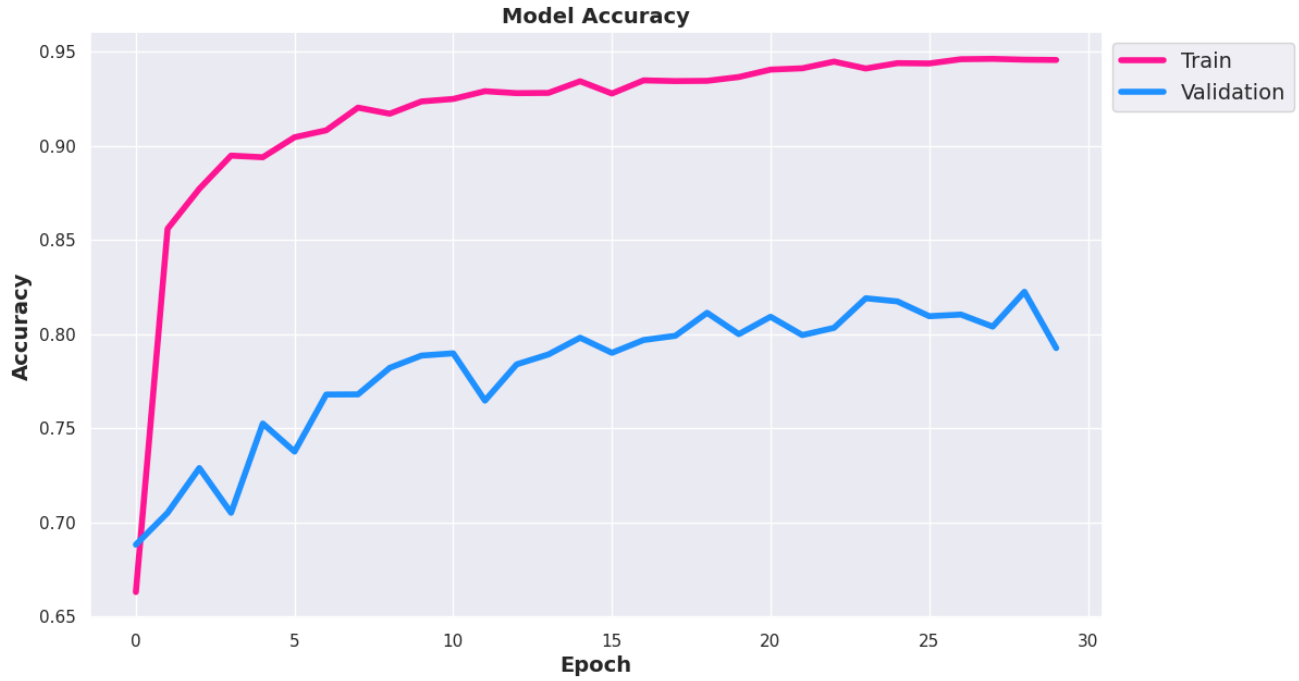
Model Training Execution Time: 32 mins 05 sec (1925 sec)

```
# Store model.fit results in a variable
data_aug_history = data_aug_model_history

# Save as DataFrame:
data_aug_history_df = pd.DataFrame(data_aug_history.history)
data_aug_history_df
```

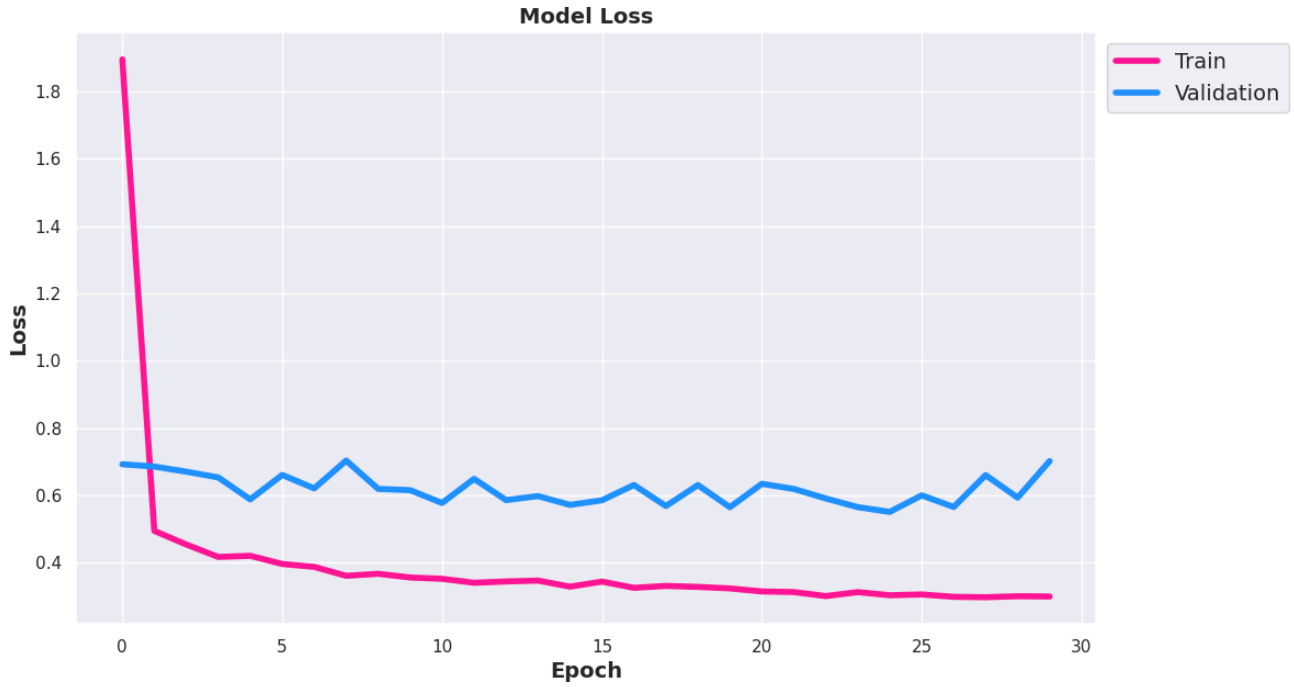
	loss	auc	val_loss	val_auc
0	1.895668	0.662861	0.692537	0.688049
1	0.494542	0.855991	0.685641	0.704987
2	0.454076	0.877225	0.670440	0.728779
3	0.416999	0.894869	0.653329	0.704987
4	0.420432	0.893995	0.588085	0.752443
5	0.396142	0.904636	0.660800	0.737535
6	0.387567	0.908322	0.620870	0.767866
7	0.360937	0.920369	0.703409	0.767949
8	0.367007	0.917144	0.619252	0.782048
9	0.356004	0.923667	0.615526	0.788572
10	0.352078	0.924988	0.577363	0.789772
11	0.340263	0.929110	0.649095	0.764635
12	0.344182	0.928074	0.586026	0.783903
13	0.346774	0.928207	0.597898	0.789177
14	0.328733	0.934398	0.571674	0.798072
15	0.343778	0.927867	0.585370	0.790035
16	0.325225	0.934864	0.630885	0.796841
17	0.330909	0.934484	0.568246	0.799084
18	0.328007	0.934619	0.630685	0.811295
19	0.323556	0.936649	0.564794	0.799968
20	0.314524	0.940581	0.634371	0.809208
21	0.312702	0.941287	0.619084	0.799461
22	0.300745	0.944839	0.590839	0.803311
23	0.312509	0.941175	0.565071	0.819040
24	0.303227	0.944033	0.550910	0.817357
25	0.305554	0.943867	0.600071	0.809506
26	0.298567	0.946114	0.565493	0.810375
27	0.297333	0.946312	0.660202	0.803968
28	0.300304	0.945856	0.593249	0.822488
29	0.299472	0.945707	0.701965	0.792564

Model Performance - Validation Accuracy

```
# Plot the model accuracy

plt.figure(figsize=(12,7))
plt.plot(data_aug_history_df['auc'], color='deeppink', linewidth=4)
plt.plot(data_aug_history_df['val_auc'], color='dodgerblue', linewidth=4)
plt.title('Model Accuracy', fontsize=14, fontweight='bold')
plt.ylabel('Accuracy', fontsize=14, fontweight='bold')
plt.xlabel('Epoch', fontsize=14, fontweight='bold')
plt.legend(['Train', 'Validation'], loc='upper left', bbox_to_anchor=(1,1), fonts
plt.show()
```



```
# Plot model loss

plt.figure(figsize=(12,7))
plt.plot(data_aug_history_df['loss'], color='deeppink', linewidth=4)
plt.plot(data_aug_history_df['val_loss'], color='dodgerblue', linewidth=4)
plt.title('Model Loss', fontsize=14, fontweight='bold')
plt.ylabel('Loss', fontsize=14, fontweight='bold')
plt.xlabel('Epoch', fontsize=14, fontweight='bold')
plt.legend(['Train', 'Validation'], loc='upper left', bbox_to_anchor=(1,1), fonts
plt.show()
```




```
# Load the TensorBoard notebook extension.

%reload_ext tensorboard
```

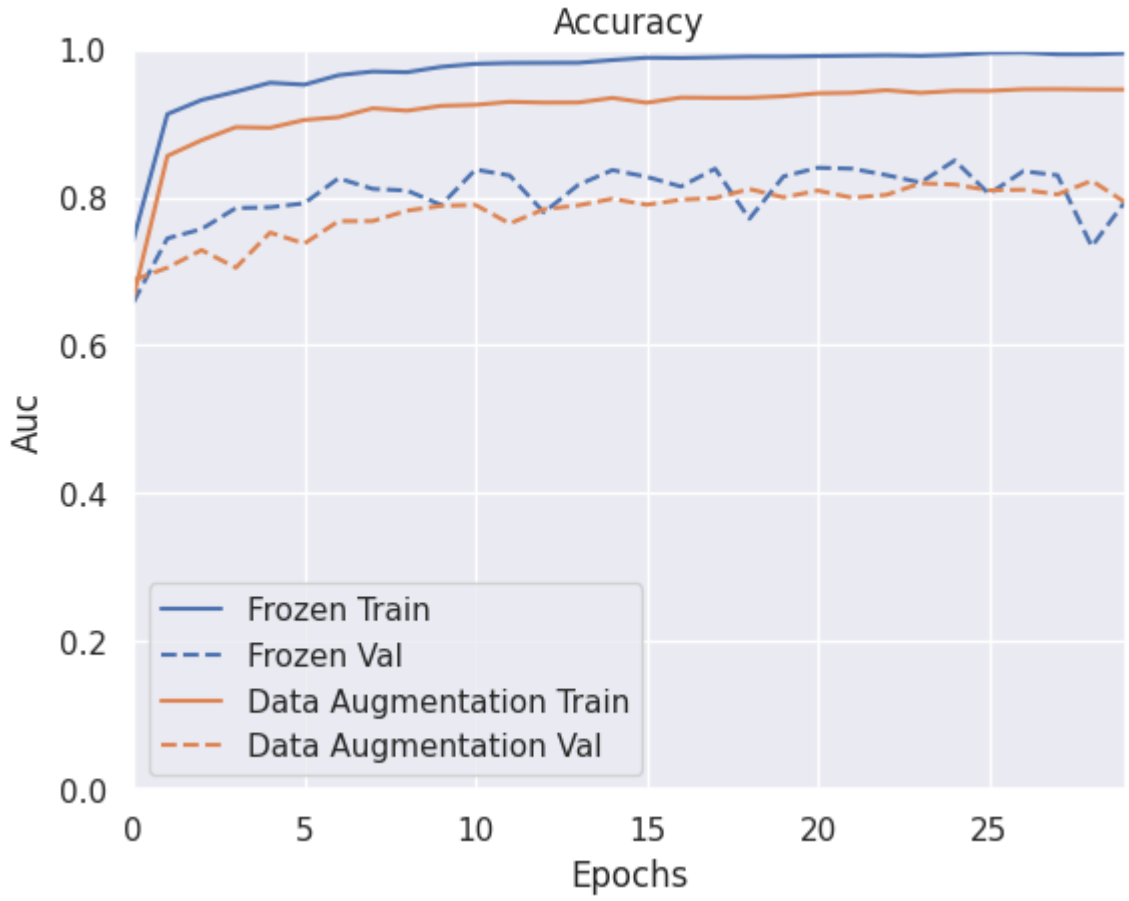
```
# Launch TensorBoard and navigate to the Profile tab to view performance profile

%tensorboard --logdir=logs
```

Compare the Performance of the 2 Models - Validation Accuracy

```
%pip install git+https://github.com/tensorflow/docs
```

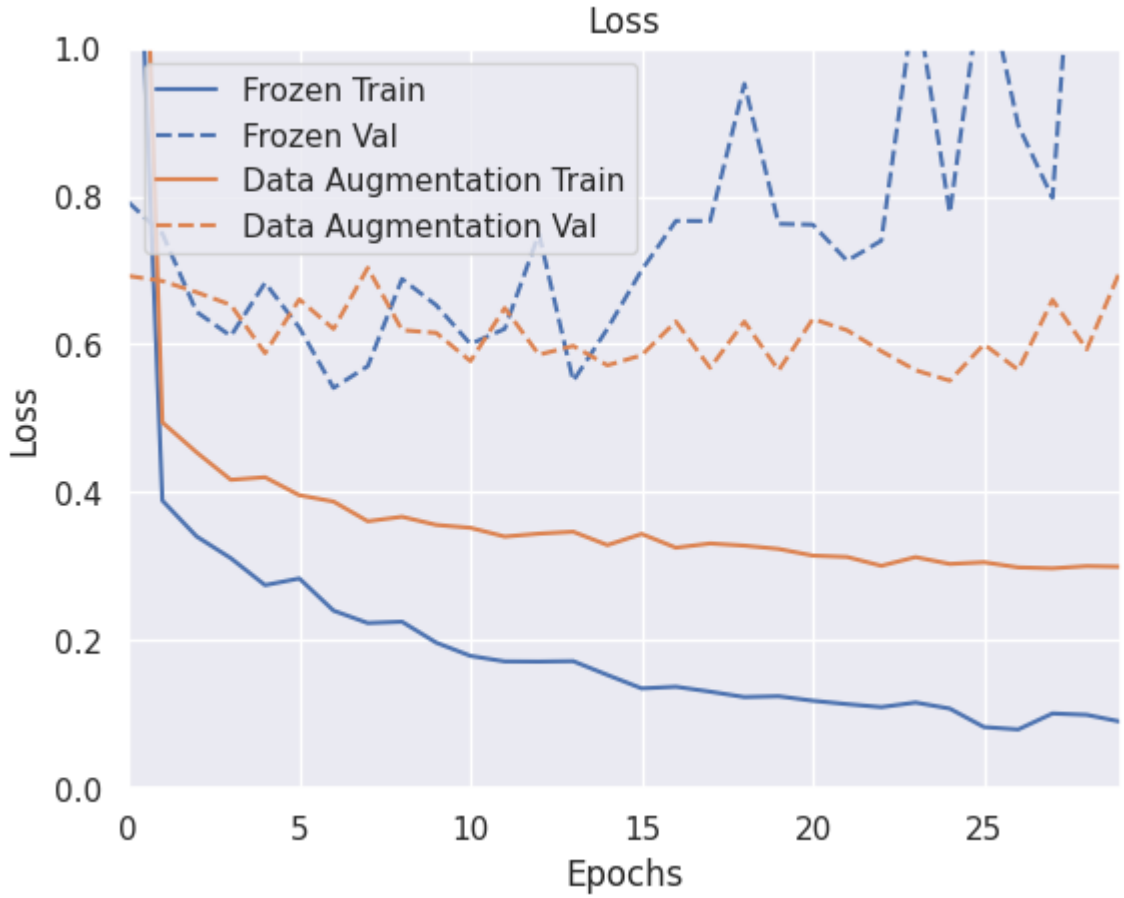
```
import tensorflow_docs as tfdocs
import tensorflow_docs.plots

plotter = tfdocs.plots.HistoryPlotter()
plotter.plot({"Frozen": model_history, "Data Augmentation": data_aug_model_histor
plt.title("Accuracy")
plt.ylim([0,1])
```

```
(0.0, 1.0)
```




```
plotter = tfdocs.plots.HistoryPlotter()
plotter.plot({"Frozen": model_history, "Data Augmentation": data_aug_model_histor
plt.title("Loss")
plt.ylim([0,1])
```

```
(0.0, 1.0)
```



Model Performance - Evaluation Accuracy

```
# Evaluate overall loss and accuracy for test data

evaluate_loss, evaluate_accuracy = model.evaluate(test_ds)

print(f"The accuracy of model evaluation was {evaluate_accuracy}.")
print(f"The loss of model evaluation was {evaluate_loss}.")
```

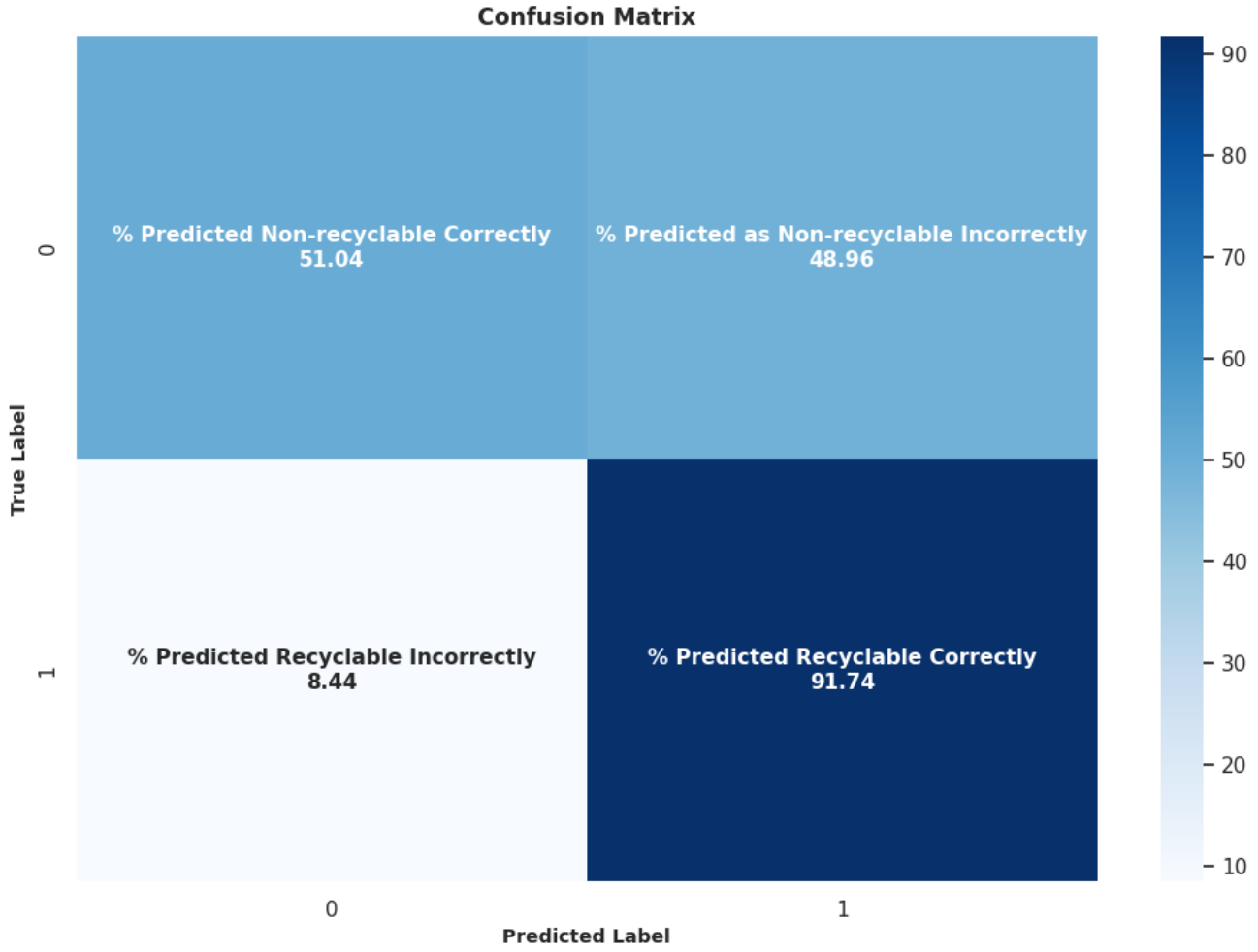
```
17/17 [==============================] - 5s 224ms/step - loss: 0.6216 - auc: (
The accuracy of model evaluation was 0.8065441846847534.
The loss of model evaluation was 0.6215570569038391.
```

```
# Calculate confusion matrix

Y_pred = model.predict(test_ds, test_ds_sample_num // prediction_batch_size + 1)
y_pred = np.argmax(Y_pred, axis=1)
cm = confusion_matrix(test_ds.classes, y_pred)
print('Confusion Matrix')
print(cm)
```

```
17/17 [==============================] - 5s 234ms/step
Confusion Matrix
[[271 260]
 [ 45 489]]
```

```
# Converting confusion matrix to percentage

perc1 = round(cm[0,0] / test_n_num * 100, 2)
perc2 = round(cm[0,1] / test_n_num * 100, 2)
perc3 = round(cm[1,0] / test_r_num * 100, 2)
perc4 = round(cm[1,1] / test_r_num * 100, 2)

# Put calculated numbers into a 2D array

cm_data = np.array([[perc1, perc2],
                [perc3, perc4]])

text = np.array([['% Predicted Non-recyclable Correctly', '% Predicted as Non-rec
                ['% Predicted Recyclable Incorrectly', '% Predicted Recyclable Co

# Combine text with values

formatted_text = (np.asarray(["{0}\n{1:.2f}".format(
text, cm_data) for text, cm_data in zip(text.flatten(), cm_data.flatten())])).res

# Plot heatmap of the confusion matrix

fig, ax = plt.subplots(figsize=(12,8))
sns.set(font_scale=1.0)
ax = sns.heatmap(cm_data, annot=formatted_text, fmt="", cmap='Blues', annot_kws={
ax.set_title("Confusion Matrix", fontsize=12, fontweight='bold')
ax.set_xlabel("Predicted Label", fontsize=10, fontweight='bold')
ax.set_ylabel("True Label", fontsize=10, fontweight='bold')
```

```
Text(116.24999999999999, 0.5, 'True Label')
```




```
# Test Case 1 on Non-recyclable

test_non_recyclable_1 = load_img(test_folder + '/Non-recyclable' + '/disposable-f
classPrediction(test_non_recyclable_1)

# Correct prediction result
```

```
1/1 [==============================] - 0s 29ms/step
The image belongs to Non-recycle waste class, probability: 0.9536254405975342
```

```
# Test Case 2 on Non-recyclable

test_non_recyclable_2 = load_img(test_folder + '/Non-recyclable' + '/disposable-p
classPrediction(test_non_recyclable_2)

# Correct prediction result
```

```
1/1 [==============================] - 0s 26ms/step
The image belongs to Non-recycle waste class, probability: 0.9901735782623291
```

```
# Test Case 3 on Non-recyclable

test_non_recyclable_3 = load_img(test_folder + '/Non-recyclable' + '/non-recyclab
classPrediction(test_non_recyclable_3)

# Incorrect prediction result
```

```
1/1 [==============================] - 0s 85ms/step
The image belongs to Recycle waste class, probability: 0.6794881224632263.
```

```
# Test Case 4 on Non-recyclable

test_non_recyclable_4 = load_img(test_folder + '/Non-recyclable' + '/non-recyclab
classPrediction(test_non_recyclable_4)

# Incorrect prediction result
```

```
1/1 [==============================] - 0s 29ms/step
The image belongs to Recycle waste class, probability: 0.7297237515449524.
```