BANA274 Deep Learning Project

Group 2 Project: Potato Chips Quality Control

Team Members: Mehak Katra, Aysouda Vatanparast, Kentrick Kepawitono, Urvi Vaidya, Moiz Nawab, Shahin Chinichian Moghaddam

```
#Setting up the model
from google.colab import drive
drive.mount('gdrive')
```

```
     Mounted at gdrive
```

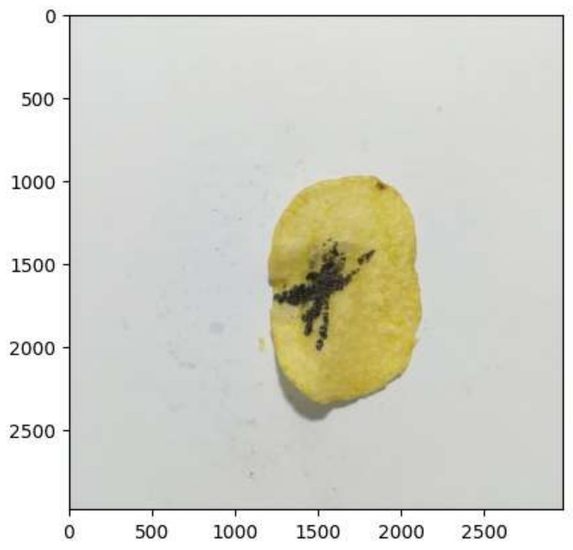
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy
from PIL import Image
from scipy import ndimage
import os

%matplotlib inline
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout, BatchNormalization
from tensorflow.keras import regularizers
from tensorflow.keras.callbacks import Callback,ModelCheckpoint, EarlyStopping
from tensorflow.keras.preprocessing import image
from tensorflow.keras.initializers import RandomNormal, he_normal, GlorotNormal
from sklearn.metrics import accuracy_score
from tensorflow.keras import optimizers
```

```
import tensorflow as tf
print(tf.__version__)
```

```
     2.12.0
```

```
orig_im = Image.open('gdrive//My Drive//Colab Notebooks//Dataset//Train//Defective Chips//IMG_20210319_010422.jpg')
plt.imshow(orig_im)
plt.show()
```

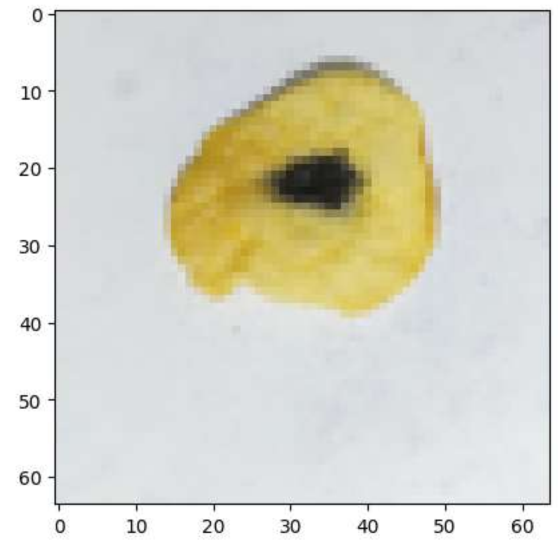


```
#Assign training set with labels
defective_train_dir = "gdrive//My Drive//Colab Notebooks//Dataset//Augmented Training Set//Defected Chips"
non_defective_train_dir = "gdrive//My Drive//Colab Notebooks//Dataset//Augmented Training Set//Non-defect Chips"
```

```
X_train_orig = []
Y_train_orig = []
```

```
image_size = (64, 64)

# Load the defective training images and assign a label of 0
for filename in os.listdir(defective_train_dir):
    if filename.endswith(".jpg"):
        filepath = os.path.join(defective_train_dir, filename)
        image = Image.open(filepath)
        image = image.resize(image_size)
        image = np.array(image)
        X_train_orig.append(image)
        Y_train_orig.append(0)

# Load the non-defective training images and assign a label of 1
for filename in os.listdir(non_defective_train_dir):
    if filename.endswith(".jpg"):
        filepath = os.path.join(non_defective_train_dir, filename)
        image = Image.open(filepath)
        image = image.resize(image_size) #original size = (2500,2500)
        image = np.array(image)
        X_train_orig.append(image)
        Y_train_orig.append(1)

X_train_orig = np.array(X_train_orig)
Y_train_orig = np.array(Y_train_orig)


# Load the test data

# Define the paths to the directories with the test images
defective_test_dir = "gdrive//My Drive//Colab Notebooks//Dataset//Augmented Test Set//Defect Chips"
non_defective_test_dir = "gdrive//My Drive//Colab Notebooks//Dataset//Augmented Test Set//No defect Chips"

#Creating divisions for defective and non-defective images
filenames1 = os.listdir(defective_test_dir)
test_size1 = len(filenames1) // 2

filenames2 = os.listdir(non_defective_test_dir)
test_size2 = len(filenames2) // 2

test_filenames1 = filenames1[:test_size1]
crossval_filenames1 = filenames1[test_size1:]

test_filenames2 = filenames2[:test_size2]
crossval_filenames2 = filenames2[test_size2:]

X_test_orig = []
Y_test_orig = []
X_cv_orig = []
Y_cv_orig = []

image_size = (64, 64)

# Load the defective test & cross-validation images and assign a label of 0
for filename in test_filenames1:
    if filename.endswith(".jpg"):
        filepath = os.path.join(defective_test_dir, filename)
        image = Image.open(filepath)
        image = image.resize(image_size)
        image = np.array(image)
        X_test_orig.append(image)
        Y_test_orig.append(0)

for filename in crossval_filenames1:
    if filename.endswith(".jpg"):
        filepath = os.path.join(defective_test_dir, filename)
        image = Image.open(filepath)
        image = image.resize(image_size)
        image = np.array(image)
        X_cv_orig.append(image)
        Y_cv_orig.append(0)

# Load the non-defective test images and assign a label of 1
for filename in test_filenames2:
    if filename.endswith(".jpg"):
        filepath = os.path.join(non_defective_test_dir, filename)
        image = Image.open(filepath)
        image = image.resize(image_size)
        image = np.array(image)
        X_test_orig.append(image)
        Y_test_orig.append(1)

for filename in crossval_filenames2:
    if filename.endswith(".jpg"):
        filepath = os.path.join(non_defective_test_dir, filename)
        image = Image.open(filepath)
        image = image.resize(image_size)
        image = np.array(image)
        X_cv_orig.append(image)
        Y_cv_orig.append(1)

X_test_orig = np.array(X_test_orig)
Y_test_orig = np.array(Y_test_orig)

X_cv_orig = np.array(X_cv_orig)
Y_cv_orig = np.array(Y_cv_orig)


index=40
plt.imshow(X_train_orig[index])
if Y_train_orig[index] == 1:
    print ("y = " + str(Y_train_orig[index]) + ", it's a non-defective potato chip.")
else:
    print ("y = " + str(Y_train_orig[index]) + ", it's a defective potato chip.")
```

```
y = 0, it's a defective potato chip.
```



```
m_train = X_train_orig.shape[0]
m_test = X_test_orig.shape[0]
m_cv = X_cv_orig.shape[0]
num_px = X_train_orig.shape[1]


print ("Number of training examples: m_train = " + str(m_train))
print ("Number of testing examples: m_test = " + str(m_test))
print ("Number of cross-validation examples: m_cv = " + str(m_cv))
print ("Height/Width of each image: num_px = " + str(num_px))
print ("Each image is of size: (" + str(num_px) + ", " + str(num_px) + ", 3)")
print ("train_set_x shape: " + str(X_train_orig.shape))
print ("train_set_y shape: " + str(Y_train_orig.shape))
print ("test_set_x shape: " + str(X_test_orig.shape))
print ("test_set_y shape: " + str(Y_test_orig.shape))
print ("cv_set_x shape: " + str(X_cv_orig.shape))
print ("cv_set_y shape: " + str(Y_cv_orig.shape))
```

```
Number of training examples: m_train = 8469
Number of testing examples: m_test = 384
Number of cross-validation examples: m_cv = 384
Height/Width of each image: num_px = 64
Each image is of size: (64, 64, 3)
train_set_x shape: (8469, 64, 64, 3)
train_set_y shape: (8469,)
test_set_x shape: (384, 64, 64, 3)
test_set_y shape: (384,)
```

```
cv_set_x shape: (384, 64, 64, 3)
cv_set_y shape: (384,)
```

```
# Reshape the training and test examples
train_set_x_flatten = X_train_orig.reshape(X_train_orig.shape[0],-1)
test_set_x_flatten = X_test_orig.reshape(X_test_orig.shape[0],-1)
cv_set_x_flatten = X_cv_orig.reshape(X_cv_orig.shape[0],-1)


print ("train_set_x_flatten shape: " + str(train_set_x_flatten.shape))
print ("train_set_y shape: " + str(Y_train_orig.shape))
print ("test_set_x_flatten shape: " + str(test_set_x_flatten.shape))
print ("test_set_y shape: " + str(Y_test_orig.shape))
print ("cv_set_x_flatten shape: " + str(cv_set_x_flatten.shape))
print ("cv_set_y shape: " + str(Y_cv_orig.shape))
print ("sanity check after reshaping: " + str(train_set_x_flatten[0:5,0]))
```

```
train_set_x_flatten shape: (8469, 12288)
train_set_y shape: (8469,)
test_set_x_flatten shape: (384, 12288)
test_set_y shape: (384,)
cv_set_x_flatten shape: (384, 12288)
cv_set_y shape: (384,)
sanity check after reshaping: [217 222 214 216 223]
```

```
print ("train_set_x shape: " + str((train_set_x_flatten.shape)))
print ("train_set_y shape: " + str(Y_train_orig.shape))
print ("test_set_x shape: " + str(test_set_x_flatten.shape))
print ("test_set_y shape: " + str(Y_test_orig.shape))
print ("cv_set_x shape: " + str(cv_set_x_flatten.shape))
print ("cv_set_y shape: " + str(Y_cv_orig.shape))
```

```
train_set_x shape: (8469, 12288)
train_set_y shape: (8469,)
test_set_x shape: (384, 12288)
test_set_y shape: (384,)
cv_set_x shape: (384, 12288)
cv_set_y shape: (384,)
```

```
#Standardizing dataset
train_set_x = train_set_x_flatten/255.
test_set_x = test_set_x_flatten/255.
cv_set_x = cv_set_x_flatten/255.


def get_test_accuracy(model, x_test, y_test):
    test_loss, test_acc = model.evaluate(x=x_test, y=y_test, verbose=0)
    print('accuracy: {acc:0.3f}'.format(acc=test_acc))
```

Sequential Neural Network Model

```
model = Sequential([
                    Dense(44, activation = 'leaky_relu', input_shape = (train_set_x.shape[1],)),
                    Dense(25, activation = 'leaky_relu'),
                    Dense(1, activation = 'sigmoid')
])

# compile the model
model.compile(optimizer=optimizers.legacy.Adam(learning_rate=0.00267),
              loss = tf.keras.losses.BinaryCrossentropy(from_logits=True),
              metrics = [tf.keras.metrics.BinaryAccuracy(threshold=0.5)])
```

Saving best weights of NN model

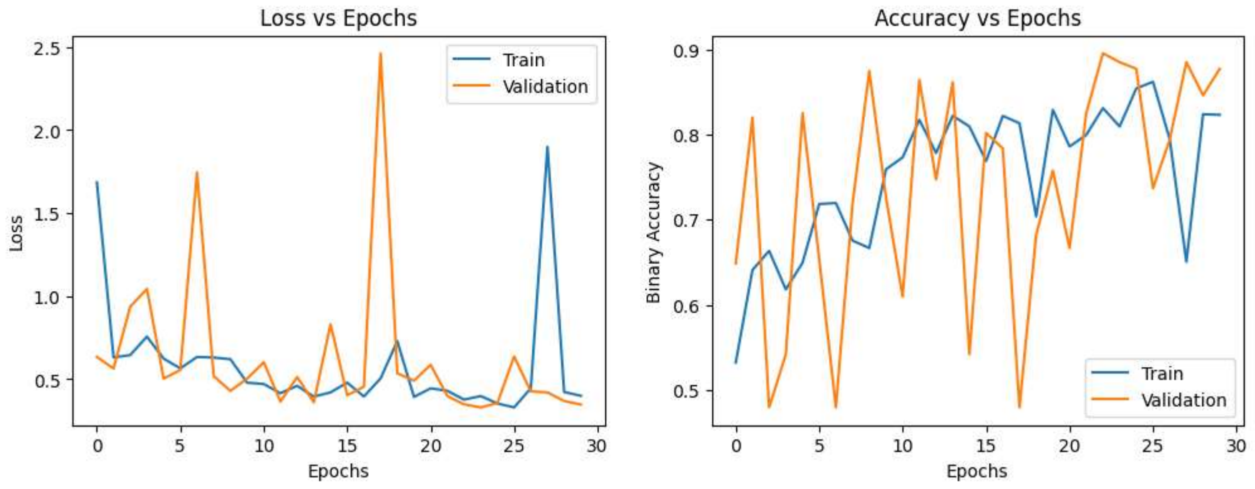
```
early_stopping = EarlyStopping(monitor='val_binary_accuracy', patience=5, restore_best_weights=True)

#checkpoint

checkpoint_path = 'model_checkpoints/best_nn_weights'
checkpoint = ModelCheckpoint(filepath=checkpoint_path,
                             save_frequency='epoch',
                             save_weights_only=True,
                             monitor='val binary accuracy',
                             save_best_only=True,
                             verbose=1)

# fit the model
history = model.fit(train_set_x, Y_train_orig, epochs = 30, batch_size = 64, verbose = 2, callbacks=[checkpoint],validation_data=(cv_set_x, Y
```

```
133/133 - 2s - loss: 0.4792 - binary_accuracy: 0.7692 - val_loss: 0.4032 - val_binary_accuracy: 0.8021 - 2s/epoch - 15ms/step
Epoch 17/30

Epoch 17: val_binary_accuracy did not improve from 0.87500
133/133 - 2s - loss: 0.3956 - binary_accuracy: 0.8222 - val_loss: 0.4542 - val_binary_accuracy: 0.7839 - 2s/epoch - 14ms/step
Epoch 18/30

Epoch 18: val_binary_accuracy did not improve from 0.87500
133/133 - 1s - loss: 0.5050 - binary_accuracy: 0.8136 - val_loss: 2.4621 - val_binary_accuracy: 0.4792 - 1s/epoch - 11ms/step
Epoch 19/30

Epoch 19: val_binary_accuracy did not improve from 0.87500
133/133 - 1s - loss: 0.7294 - binary_accuracy: 0.7039 - val_loss: 0.5357 - val_binary_accuracy: 0.6823 - 1s/epoch - 10ms/step
Epoch 20/30

Epoch 20: val_binary_accuracy did not improve from 0.87500
133/133 - 1s - loss: 0.3931 - binary_accuracy: 0.8294 - val_loss: 0.4912 - val_binary_accuracy: 0.7578 - 1s/epoch - 10ms/step
Epoch 21/30

Epoch 21: val_binary_accuracy did not improve from 0.87500
133/133 - 1s - loss: 0.4448 - binary_accuracy: 0.7862 - val_loss: 0.5867 - val_binary_accuracy: 0.6667 - 1s/epoch - 11ms/step
Epoch 22/30

Epoch 22: val_binary_accuracy did not improve from 0.87500
133/133 - 2s - loss: 0.4302 - binary_accuracy: 0.7999 - val_loss: 0.3984 - val_binary_accuracy: 0.8255 - 2s/epoch - 17ms/step
Epoch 23/30

Epoch 23: val_binary_accuracy improved from 0.87500 to 0.89583, saving model to model_checkpoints/best_nn_weights
133/133 - 2s - loss: 0.3770 - binary_accuracy: 0.8314 - val_loss: 0.3484 - val_binary_accuracy: 0.8958 - 2s/epoch - 19ms/step
Epoch 24/30

Epoch 24: val_binary_accuracy did not improve from 0.89583
133/133 - 2s - loss: 0.3984 - binary_accuracy: 0.8098 - val_loss: 0.3301 - val_binary_accuracy: 0.8854 - 2s/epoch - 18ms/step
Epoch 25/30

Epoch 25: val_binary_accuracy did not improve from 0.89583
133/133 - 3s - loss: 0.3538 - binary_accuracy: 0.8545 - val_loss: 0.3559 - val_binary_accuracy: 0.8776 - 3s/epoch - 23ms/step
Epoch 26/30

Epoch 26: val_binary_accuracy did not improve from 0.89583
133/133 - 2s - loss: 0.3300 - binary_accuracy: 0.8623 - val_loss: 0.6367 - val_binary_accuracy: 0.7370 - 2s/epoch - 13ms/step
Epoch 27/30

Epoch 27: val_binary_accuracy did not improve from 0.89583
133/133 - 3s - loss: 0.4458 - binary_accuracy: 0.7949 - val_loss: 0.4267 - val_binary_accuracy: 0.7943 - 3s/epoch - 20ms/step
Epoch 28/30

Epoch 28: val_binary_accuracy did not improve from 0.89583
133/133 - 3s - loss: 1.8986 - binary_accuracy: 0.6506 - val_loss: 0.4202 - val_binary_accuracy: 0.8854 - 3s/epoch - 20ms/step
Epoch 29/30

Epoch 29: val_binary_accuracy did not improve from 0.89583
133/133 - 3s - loss: 0.4226 - binary_accuracy: 0.8242 - val_loss: 0.3683 - val_binary_accuracy: 0.8464 - 3s/epoch - 19ms/step
Epoch 30/30

Epoch 30: val_binary_accuracy did not improve from 0.89583
133/133 - 3s - loss: 0.4000 - binary_accuracy: 0.8235 - val_loss: 0.3471 - val_binary_accuracy: 0.8776 - 3s/epoch - 24ms/step
```

Loss and metric(s) graph of NN model

```
# Plot the training and validation loss
frame = pd.DataFrame(history.history)
epochs = np.arange(len(frame))

fig = plt.figure(figsize=(12,4))

# Loss plot
ax = fig.add_subplot(121)
ax.plot(epochs, frame['loss'], label="Train")
ax.plot(epochs, frame['val_loss'], label="Validation")
ax.set_xlabel("Epochs")
ax.set_ylabel("Loss")
ax.set_title("Loss vs Epochs")
ax.legend()

# Accuracy plot
ax = fig.add_subplot(122)
ax.plot(epochs, frame['binary_accuracy'], label="Train")
ax.plot(epochs, frame['val_binary_accuracy'], label="Validation")
ax.set_xlabel("Epochs")
ax.set_ylabel("Binary Accuracy")
ax.set_title("Accuracy vs Epochs")
ax.legend()
```

```
    <matplotlib.legend.Legend at 0x7f83d9e05810>
```



```
#test accuracy
get_test_accuracy(model, test_set_x, Y_test_orig)
```

```
    /usr/local/lib/python3.10/dist-packages/keras/backend.py:5703: UserWarning: "`binary_crossentropy` received `from_logits=True`, but the
      output, from_logits = _get_logits(
    accuracy: 0.870
```

```
#cross-val accuracy
get_test_accuracy(model, cv_set_x, Y_cv_orig)
```

```
    accuracy: 0.878
```

CNN model

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, BatchNormalization, Dropout, Conv2D, MaxPooling2D
from tensorflow.keras import regularizers

model1 = Sequential([
                    Conv2D(filters=5, kernel_size=(3,3), strides=(1,1), padding='valid', activation = 'relu', input_shape=(64,64,3)),
                    MaxPooling2D(pool_size=(2,2), strides=(1,1), padding='valid'),
                    Conv2D(filters=5, kernel_size=(3,3), strides=(1,1), padding='valid', activation = 'relu'),
                    MaxPooling2D(pool_size=(2,2), strides=(1,1), padding='valid'),
                    Flatten(),
                    Dense(120, activation = 'leaky_relu',kernel_regularizer=tf.keras.regularizers.l2(0.001)),
                    BatchNormalization(),
                    Dense(75, activation='leaky_relu',kernel_regularizer=tf.keras.regularizers.l2(0.001)),
                    BatchNormalization(),
                    Dense(1, activation='sigmoid', kernel_regularizer=tf.keras.regularizers.l2(0.001))
])
```

```
model1.summary()
```

```
    Model: "sequential_1"
    _________________________________________________________________
     Layer (type)                Output Shape              Param #
    =================================================================
     conv2d (Conv2D)             (None, 62, 62, 5)         140
```

```
 max_pooling2d (MaxPooling2D  (None, 61, 61, 5)        0
 )

 conv2d_1 (Conv2D)           (None, 59, 59, 5)         230

 max_pooling2d_1 (MaxPooling  (None, 58, 58, 5)        0
 2D)

 flatten (Flatten)           (None, 16820)             0

 dense_3 (Dense)             (None, 120)               2018520

 batch_normalization (BatchN  (None, 120)              480
 ormalization)

 dense_4 (Dense)             (None, 75)                9075

 batch_normalization_1 (Batc  (None, 75)               300
 hNormalization)

 dense_5 (Dense)             (None, 1)                 76

=================================================================
Total params: 2,028,821
Trainable params: 2,028,431
Non-trainable params: 390
_________________________________________________________________
```

```
model1.compile(optimizer=optimizers.legacy.Adam(learning_rate=0.00267),
              loss = tf.keras.losses.BinaryCrossentropy(from_logits=False),
              metrics = [tf.keras.metrics.BinaryAccuracy(threshold=0.5)])
```

Saving the best weights for CNN model after training

```
early_stopping = EarlyStopping(monitor='val_binary_accuracy', patience=5, restore_best_weights=True)

checkpoint_path_cnn = 'model_checkpoints/checkpoint_CNN'
checkpoint = ModelCheckpoint(filepath=checkpoint_path_cnn,
                             save_frequency='epoch',
                             save_weights_only=True,
                             monitor='val_binary_accuracy',
                             save_best_only=True,
                             verbose=1)
```

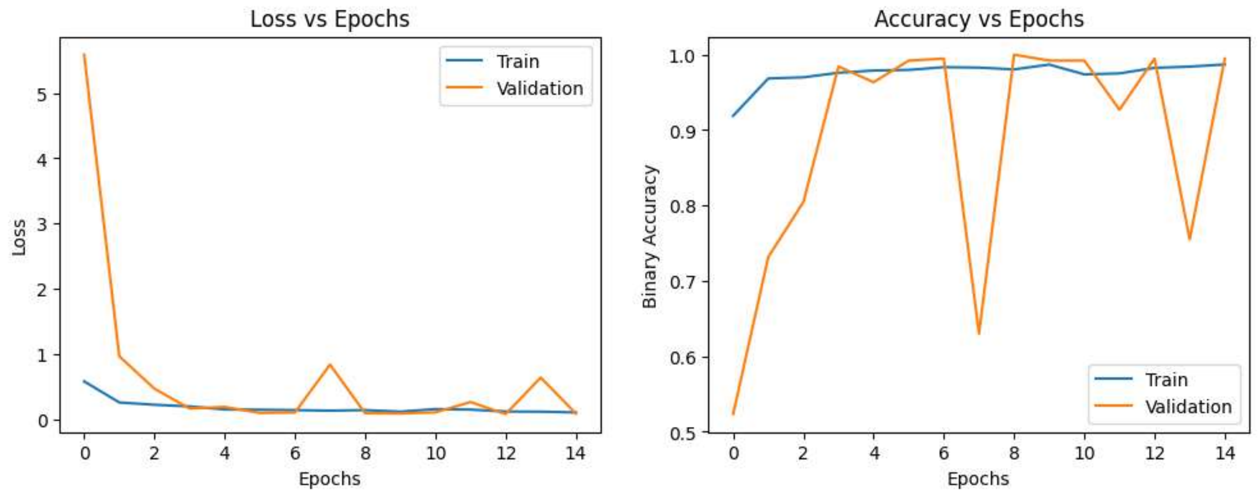
```
cnn_history = model1.fit(X_train_orig, Y_train_orig, epochs = 15, batch_size = 64, verbose = 2, callbacks=[checkpoint],validation_data=(X_cv_
```

```
Epoch 1: val_binary_accuracy improved from -inf to 0.52344, saving model to model_checkpoints/checkpoint_CNN
133/133 - 46s - loss: 0.5806 - binary_accuracy: 0.9190 - val_loss: 5.5899 - val_binary_accuracy: 0.5234 - 46s/epoch - 342ms/step
Epoch 2/15

Epoch 2: val_binary_accuracy improved from 0.52344 to 0.73177, saving model to model_checkpoints/checkpoint_CNN
133/133 - 41s - loss: 0.2571 - binary_accuracy: 0.9685 - val_loss: 0.9651 - val_binary_accuracy: 0.7318 - 41s/epoch - 305ms/step
Epoch 3/15

Epoch 3: val_binary_accuracy improved from 0.73177 to 0.80469, saving model to model_checkpoints/checkpoint_CNN
133/133 - 53s - loss: 0.2224 - binary_accuracy: 0.9700 - val_loss: 0.4685 - val_binary_accuracy: 0.8047 - 53s/epoch - 396ms/step
Epoch 4/15

Epoch 4: val_binary_accuracy improved from 0.80469 to 0.98438, saving model to model_checkpoints/checkpoint_CNN
133/133 - 50s - loss: 0.1966 - binary_accuracy: 0.9760 - val_loss: 0.1645 - val_binary_accuracy: 0.9844 - 50s/epoch - 373ms/step
Epoch 5/15

Epoch 5: val_binary_accuracy did not improve from 0.98438
133/133 - 56s - loss: 0.1498 - binary_accuracy: 0.9790 - val_loss: 0.1886 - val_binary_accuracy: 0.9635 - 56s/epoch - 425ms/step
Epoch 6/15

Epoch 6: val_binary_accuracy improved from 0.98438 to 0.99219, saving model to model_checkpoints/checkpoint_CNN
133/133 - 47s - loss: 0.1459 - binary_accuracy: 0.9798 - val_loss: 0.0971 - val_binary_accuracy: 0.9922 - 47s/epoch - 351ms/step
Epoch 7/15

Epoch 7: val_binary_accuracy improved from 0.99219 to 0.99479, saving model to model_checkpoints/checkpoint_CNN
133/133 - 54s - loss: 0.1399 - binary_accuracy: 0.9835 - val_loss: 0.1031 - val_binary_accuracy: 0.9948 - 54s/epoch - 409ms/step
Epoch 8/15

Epoch 8: val_binary_accuracy did not improve from 0.99479
Epoch 9: val_binary_accuracy improved from 0.99479 to 1.00000, saving model to model_checkpoints/checkpoint_CNN
133/133 - 55s - loss: 0.1397 - binary_accuracy: 0.9805 - val_loss: 0.0916 - val_binary_accuracy: 1.0000 - 55s/epoch - 415ms/step
Epoch 10/15

Epoch 10: val_binary_accuracy did not improve from 1.00000
133/133 - 54s - loss: 0.1164 - binary_accuracy: 0.9869 - val_loss: 0.0888 - val_binary_accuracy: 0.9922 - 54s/epoch - 409ms/step
Epoch 11/15

Epoch 11: val_binary_accuracy did not improve from 1.00000
133/133 - 53s - loss: 0.1550 - binary_accuracy: 0.9739 - val_loss: 0.1043 - val_binary_accuracy: 0.9922 - 53s/epoch - 399ms/step
Epoch 12/15

Epoch 12: val_binary_accuracy did not improve from 1.00000
133/133 - 46s - loss: 0.1479 - binary_accuracy: 0.9752 - val_loss: 0.2643 - val_binary_accuracy: 0.9271 - 46s/epoch - 342ms/step
Epoch 13/15

Epoch 13: val_binary_accuracy did not improve from 1.00000
133/133 - 41s - loss: 0.1187 - binary_accuracy: 0.9825 - val_loss: 0.0783 - val_binary_accuracy: 0.9948 - 41s/epoch - 309ms/step
Epoch 14/15

Epoch 14: val_binary_accuracy did not improve from 1.00000
133/133 - 41s - loss: 0.1158 - binary_accuracy: 0.9842 - val_loss: 0.6392 - val_binary_accuracy: 0.7552 - 41s/epoch - 309ms/step
Epoch 15/15

Epoch 15: val_binary_accuracy did not improve from 1.00000
133/133 - 41s - loss: 0.1049 - binary_accuracy: 0.9870 - val_loss: 0.0873 - val_binary_accuracy: 0.9948 - 41s/epoch - 308ms/step
```

Loss and metric(s) graphs

```
# Plot the training and validation loss
frame = pd.DataFrame(cnn_history.history)
epochs = np.arange(len(frame))

fig = plt.figure(figsize=(12,4))

# Loss plot
ax = fig.add_subplot(121)
ax.plot(epochs, frame['loss'], label="Train")
ax.plot(epochs, frame['val_loss'], label="Validation")
ax.set_xlabel("Epochs")
ax.set_ylabel("Loss")
ax.set_title("Loss vs Epochs")
ax.legend()

# Accuracy plot
ax = fig.add_subplot(122)
ax.plot(epochs, frame['binary_accuracy'], label="Train")
ax.plot(epochs, frame['val_binary_accuracy'], label="Validation")
ax.set_xlabel("Epochs")
ax.set_ylabel("Binary Accuracy")
ax.set_title("Accuracy vs Epochs")
ax.legend()
```

```
<matplotlib.legend.Legend at 0x7f841b8cc7f0>
```



```
#test accuracy
get_test_accuracy(model1, X_test_orig, Y_test_orig)
```

```
     accuracy: 0.964
```

```
#test accuracy
get_test_accuracy(model1, X_cv_orig, Y_cv_orig)
```

```
     accuracy: 0.995
```

Loading weights of NN and CNN model

```
# NN model weights
model = Sequential([
                    Dense(44, activation = 'leaky_relu', input_shape = (train_set_x.shape[1],)),
                    Dense(25, activation = 'leaky_relu'),
                    Dense(1, activation = 'sigmoid')
])

model.compile(optimizer=optimizers.legacy.Adam(learning_rate=0.00267),
              loss = tf.keras.losses.BinaryCrossentropy(from_logits=True),
              metrics = [tf.keras.metrics.BinaryAccuracy(threshold=0.5)])

model.load_weights(checkpoint_path)
```

```
     <tensorflow.python.checkpoint.checkpoint.CheckpointLoadStatus at 0x7f841b594ee0>
```

```
# CNN model weights
model1 = Sequential([
                    Conv2D(filters=5, kernel_size=(3,3), strides=(1,1), padding='valid', activation = 'relu', input_shape=(64,64,3)),
                    MaxPooling2D(pool_size=(2,2), strides=(1,1), padding='valid'),
                    Conv2D(filters=5, kernel_size=(3,3), strides=(1,1), padding='valid', activation = 'relu'),
                    MaxPooling2D(pool_size=(2,2), strides=(1,1), padding='valid'),
                    Flatten(),
                    Dense(120, activation = 'relu',kernel_regularizer=tf.keras.regularizers.l2(0.001)),
                    BatchNormalization(),
                    Dense(75, activation='relu',kernel_regularizer=tf.keras.regularizers.l2(0.001)),
                    BatchNormalization(),
                    Dense(1, activation='sigmoid', kernel_regularizer=tf.keras.regularizers.l2(0.001))
])

model1.compile(optimizer=optimizers.legacy.Adam(learning_rate=0.00267),
              loss = tf.keras.losses.BinaryCrossentropy(from_logits=False),
              metrics = [tf.keras.metrics.BinaryAccuracy(threshold=0.5)])

model1.load_weights(checkpoint_path_cnn)
```

```
     <tensorflow.python.checkpoint.checkpoint.CheckpointLoadStatus at 0x7f4cbaf48220>
```

Random Test

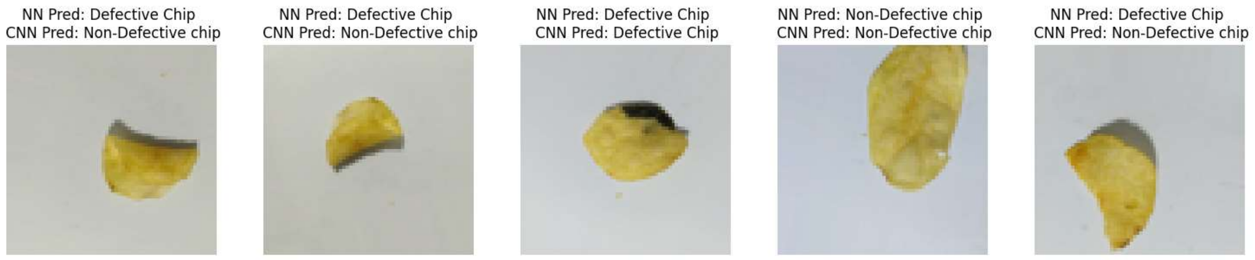
```
# Select 5 images from test set

n = np.random.randint(0,X_test_orig.shape[0],5)
df=pd.Series()
for img in X_test_orig[n]:
 pic = Image.fromarray(img)
 df.loc[len(df)] = pic
```

```
     <ipython-input-33-bd6137606c07>:4: FutureWarning: The default dtype for empty Series will be 'object' instead of 'float64' in a future v
       df=pd.Series()
```

```
selected_images = test_set_x[n]
selected_images_cnn = X_test_orig[n]
selected_labels = Y_test_orig[n]

# Use previous model to predict labels
nn_predictions = model.predict(selected_images)
nn_predictions = np.where(nn_predictions > 0.5, 1, 0)
```

```
# Use CNN model to predict labels
cnn_predictions = model1.predict(selected_images_cnn)
cnn_predictions = np.where(cnn_predictions > 0.5, 1, 0)

# Display images with predicted labels
fig, axes = plt.subplots(nrows=1, ncols=5, figsize=(15, 3))

for i, ax in enumerate(axes):
  ax.imshow(selected_images_cnn[i])
  if nn_predictions[i] == 1:
    ndchip = "Non-Defective chip"
  else:
    ndchip= "Defective Chip"
  if cnn_predictions[i] == 1:
    cnnchip = "Non-Defective chip"
  else:
    cnnchip= "Defective Chip"
  ax.set_title(f'NN Pred: {ndchip} \n CNN Pred: {cnnchip}')
  ax.axis('off')

plt.tight_layout()
plt.show()
```

```
    1/1 [==============================] - 0s 28ms/step
    1/1 [==============================] - 0s 34ms/step
```


NN Pred: Defective Chip
CNN Pred: Non-Defective chip
NN Pred: Defective Chip
CNN Pred: Non-Defective chip
NN Pred: Defective Chip
CNN Pred: Defective Chip
NN Pred: Non-Defective chip
CNN Pred: Non-Defective chip
NN Pred: Defective Chip
CNN Pred: Non-Defective chip

In our project, in the primitive model, we observed ~5% variance. We still tried to reduce this using regualrization methods, but it increased the variance and affected the bias as well, thus introducing over-fitting and under-fitting under different scenarios._

So, we also checked the performance by developing a CNN model. It gave us the best results in comparison with the human performance. The variance and bias observed here are as expected- ~1-2%. This is acceptable according to industry standards. Even if a human tries to segregate the chips in real-time, ~5% human error can be expected when manufactured on a large scale.

Thus, to conclude, our CNN model gives the best performance. In the NN model, the base model performs the best without introducing weight initialization techniques/regularization techniques.

✓ 0s completed at 12:53 PM