

## **HOMEWORK #3: Image Classification**

### **CSCI 543: Assignment 3**

#### **Problem #1: Convolutional Neural Network**

##### **Report:**

### **Building a Convnet from Scratch (Dog vs. Cat Image Classification)**

#### **Abstract**

I implement and run a tutorial (with some modification) on image classification via machine learning (ML) deep neural network. More specifically, I perform binary classification on an image dataset by convolutional neural network. This image dataset contains cats and dogs image files (jpg format) stored on disk which is unstructured data. I build a convolutional network with the following layers (i) input layer, (ii) 2D convolutional layer, (iii) 2D max pooling layer, (iv) flatten layer, and (v) dense layer. I perform this implementation of image classification using python programming language. Besides, I am using Keras, TensorFlow etc. for image preprocessing, image reshaping and building this Convolution Neural Network model for image classification. This ML ConvNet model can analyze images from the dataset and able to detect cat/dog from those images. Thus it performs binary classification to separate cat/dog from images. With 12 epochs this model shows 98% accuracy during training the model and 73% accuracy during the validation phase.

TensorFlow tutorial link: [https://colab.research.google.com/github/google/eng-edu/blob/master/ml/pc/exercises/image\\_classification\\_part1.ipynb](https://colab.research.google.com/github/google/eng-edu/blob/master/ml/pc/exercises/image_classification_part1.ipynb)

My code in google drive (Colab):

[https://colab.research.google.com/drive/12YhohJGv39n9meCMgGB6U\\_Qmm4sz3EYG?usp=sharing](https://colab.research.google.com/drive/12YhohJGv39n9meCMgGB6U_Qmm4sz3EYG?usp=sharing)

**Tools:** Python programming language, Keras, TensorFlow, Google Colab, Google Drive.

#### **Dataset Details**

The original dataset is available on the Kaggle which contains 25,000 images for cats and dogs. I am using 2000 images of cats and dogs for training purpose and 1000 images of cats and dogs for validation purpose. So, a total of 3000 images of cats and dogs are used in this work. I am using a subset from the full dataset to reduce the time during the training of this ML model. The zip file of the dataset used in this work is stored on google cloud storage system. URL of this unstructured image dataset is - [https://storage.googleapis.com/mledu-datasets/cats\\_and\\_dogs\\_filtered.zip](https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip). Machine learning model usually required three

types of dataset to perform any experimental analysis, (i) ‘train dataset’ to train the model, (ii) ‘validation dataset’ for evaluation the quality of the model, (iii) ‘test dataset’ to test the model after the model has gone through the validation process. I don’t go through testing process, that’s why; ‘test dataset’ is not required in this work. This image dataset has two type of data; train data, validation data. The distribution of the dataset is shown in the following.

```
total training cat images: 1000
total training dog images: 1000
total validation cat images: 500
total validation dog images: 500
```

## Data Preprocessing

Data preprocessing and normalization makes data suitable to fit into the ML model. I set up data generators that will read images from the source dataset and convert them to float32 tensors, and feed them (with their labels) to this convolutional network. To normalize image data, I use min-max normalization method. This method rescaled all images by 1./255 which limit the range within 0 to 1. Besides, to be consistent all image is resized to 150x150 (height x width). I am inserting 20 images at a time by setting batch size at 20.

## The ML Approach

I am performing image classification using Convolutional Neural Network that is a well-known Machine Learning (ML) approach for image data analysis. I build a Convolutional Deep Neural Network with multiple layers to train and evaluate this ML model with the image dataset. This is a ten-layer Deep Neural Network ML model. These are - One input layer, three convolution layers, three max pooling layers, one flatten layer, two dense layers. These ten layer sequentially build the classifier to classify cat and dog images. Model layers are given below.

- i. **Input Layer:** This layer takes image data as input for further analysis. All image data must be in same shape before fed into the model. In this layer each image is reshaped by 150x150x3. Here, 150x150 for the image pixels, and 3 for three color channels: R, G, B.
- ii. **Convolution Layer(2D):** This is a convolutional layer which support 2D images. I am using 3 stack of Convolution and Max pooling layer in this ML model. The first convolution layer extracts 16 filters, 2<sup>nd</sup> one extract 32 filters and last one extract 64 filters. Each of these layers consider 3x3 matrix window.
- iii. **Max Pooling Layer (2D):** Max Polling layer support 2D images. This extract maximum vector from previous layer matrix and reduce the size using 2x2 matrix window. I am using 3 max pooling layers with 3 convolution layers.
- iv. **Flatten Layer:** This layer map features from the previous layer to a 1-dimensinal tensor so that this can be connected with a fully connected layer.
- v. **Dense Layer:** Output from the previous layer piped through 512 hidden neurons.
- vi. **Dense\_1(Output) Layer:** Final single output neuron.

The sequence of these layers are illustrated in the following figure 1. The figure 1 also shows output shape of each layer and total number of trainable parameters generated from this machine learning model.

Model: "model"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 150, 150, 3)]	0
conv2d (Conv2D)	(None, 148, 148, 16)	448
max_pooling2d (MaxPooling2D)	(None, 74, 74, 16)	0
conv2d_1 (Conv2D)	(None, 72, 72, 32)	4640
max_pooling2d_1 (MaxPooling2D)	(None, 36, 36, 32)	0
conv2d_2 (Conv2D)	(None, 34, 34, 64)	18496
max_pooling2d_2 (MaxPooling2D)	(None, 17, 17, 64)	0
flatten (Flatten)	(None, 18496)	0
dense (Dense)	(None, 512)	9470464
dense_1 (Dense)	(None, 1)	513
Total params: 9,494,561		
Trainable params: 9,494,561		
Non-trainable params: 0		

Figure: Model Summary.

Two activation function is used within these layers.

- Relu** – All layers use relu activation function. Relu overcome vanishing gradient problem, learning faster and better. Relu is default for Convolutional Neural Network.
- Sigmoid** – This is a binary classification which will require ‘0’ or ‘1’ label to classify cats and dogs in the images. That’s why I am using ‘sigmoid’ activation function in this work. Only output(Dense\_1) layer use sigmoid activation function.

**RMSProp** (Root Mean Square Propagation) is used as optimization algorithm in this model. This is preferable because this automates learning rate tuning. Initial learning rate is 0.001 in this model. Other optimizer like ‘Adam’ also work equally for this model.

The complete procedure of this ML approach for Image Classification is shown below.

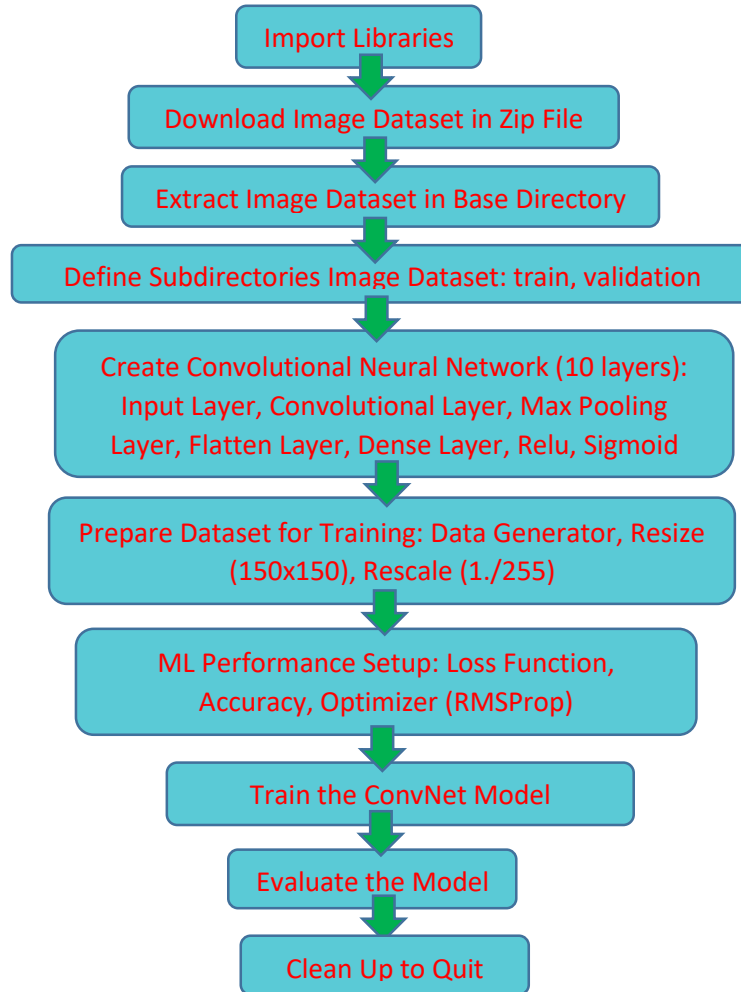


Figure 2: Complete procedure of this ConvNet ML approach for Image Classification

## ML Model Performance

To evaluate the performance of this ConvNet model, I perform loss analysis for the training and validation dataset using ‘Binary Crossentropy’ loss function and also determine the accuracy for these two datasets. Figure 3 and figure 4 present the accuracy and loss during training and validation phase. Maximum accuracy and loss from this model is given below.

Training Data Accuracy: 0.9869999885559082 (99%)  
Training Data Loss: 0.7454967498779297 (75%)  
Validation Data Accuracy 0.7200000286102295 (72%)  
Validation Data Loss 2.809699058532715 (280%)

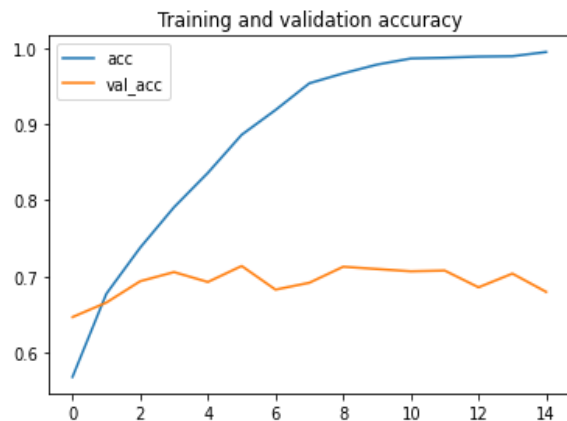


Figure 3.: Training and Validation Accuracy

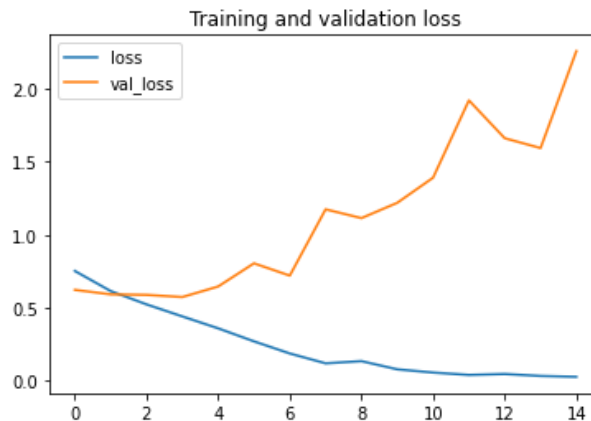


Figure 4.: Training and Validation Loss

From these figures it is clear that this ML model is “overfitting”. Figure 3 shows training accuracy of this model is very close to 99% while during validation accuracy is near to 72%. This is because this model considers some features which are irrelevant for image prediction. To overcome this “overfitting” one easy solution is train this model with more data. Some ways to overcome “overfitting” a model is given below.

- Reduce overfitting by training the network on more examples. I should consider the full 25,000 images for this model, instead of the subset (3000 images).
- Reduce overfitting by changing the complexity of the network. The complexity can be by changing number of weights (network structure) in the network. Another approach to controlling the complexity of a model is changing the value of weights in the network (network parameter).

### Codes with Comments:

I am giving explanation of each line segment from my implementation of this ConvNet neural network to perform image classification.

Colab link for the code:

[https://colab.research.google.com/drive/12YhohJGv39n9meCMgGB6U\\_Qmm4sz3EYG?usp=sharing](https://colab.research.google.com/drive/12YhohJGv39n9meCMgGB6U_Qmm4sz3EYG?usp=sharing)

""image\_classification\_CNN1.ipynb

Automatically generated by Colaboratory.

Original file is located at

[https://colab.research.google.com/drive/12YhohJGv39n9meCMgGB6U\\_Qmm4sz3EYG](https://colab.research.google.com/drive/12YhohJGv39n9meCMgGB6U_Qmm4sz3EYG)

Submitted by: Md. Saifur Rahman  
Email: [mdsaifur.rahman.1@und.edu](mailto:mdsaifur.rahman.1@und.edu)  
Date: 03/19/2021

"""

```
# include libraries/packages

import os

#import zipfile

import tensorflow as tf

from tensorflow.keras import layers

from tensorflow.keras import Model

import matplotlib.pyplot as plt

import matplotlib.image as mpimg

#from tensorflow.keras import losses

from tensorflow.keras.optimizers import RMSprop

#from tensorflow.keras import preprocessing

from tensorflow.keras.preprocessing.image import ImageDataGenerator

import numpy as np

import random

from tensorflow.keras.preprocessing.image import img_to_array, load_img


# Check tensorflow version

print(tf.version.VERSION)

print(tf.__version__)


# Download Image dataset on disk and explore the dataset

url = "https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip"


dataset = tf.keras.utils.get_file( "cats_and_dogs_filtered.zip", url, extract = True, cache_dir='.',
cache_subdir="")

base_dir = os.path.join(os.path.dirname(dataset),'cats_and_dogs_filtered')
```

Submitted by: Md. Saifur Rahman  
Email: [mdsaifur.rahman.1@und.edu](mailto:mdsaifur.rahman.1@und.edu)  
Date: 03/19/2021

# Print path for base directory

base\_dir

# base directory contains train and validation subdirectories for the training and validation datasets

# train and validation subdirectories

train\_dir = os.path.join(base\_dir, 'train')

validation\_dir = os.path.join(base\_dir, 'validation')

# Directory with training cat pictures

train\_cats\_dir = os.path.join(train\_dir, 'cats')

# Directory with training dog pictures

train\_dogs\_dir = os.path.join(train\_dir, 'dogs')

# Directory with validation cat pictures

validation\_cats\_dir = os.path.join(validation\_dir, 'cats')

# Directory with validation dog pictures

validation\_dogs\_dir = os.path.join(validation\_dir, 'dogs')

# let's see first 10 filenames in the cats and dogs train directories (file naming conventions are the same in the validation directory)

# os.listdir() - Return a list containing the names of the entries in the directory given by path. The list is in arbitrary order, and does not include the special entries.

train\_cat\_fnames = os.listdir(train\_cats\_dir)

print(train\_cat\_fnames[:10])

Submitted by: Md. Saifur Rahman  
Email: [mdsaifur.rahman.1@und.edu](mailto:mdsaifur.rahman.1@und.edu)  
Date: 03/19/2021

```
train_dog_fnames = os.listdir(train_dogs_dir)
train_dog_fnames.sort()
print(train_dog_fnames[:10])
```

# Let's find out the total number of cat and dog images in the train and validation directories

```
print('total training cat images:', len(os.listdir(train_cats_dir)))
print('total training dog images:', len(os.listdir(train_dogs_dir)))
print('total validation cat images:', len(os.listdir(validation_cats_dir)))
print('total validation dog images:', len(os.listdir(validation_dogs_dir)))
```

# Commented out IPython magic to ensure Python compatibility.

#let's take a look at a few pictures to get a better sense of what the cat and dog datasets look like.  
First, configure the matplotlib parameters

# With this inline backend, the output of plotting commands is displayed inline within frontends like the google colab, directly below the code cell that produced it.

# %matplotlib inline

# Parameters for the graph; this will output images in a 4x4 configuration

nrows = 4

ncols = 4

# Index for iterating over images

pic\_index = 0

# Display a batch of 8 cat and 8 dog pictures.

# Get the current figure.If no current figure exists, a new one is created using figure().



Submitted by: Md. Saifur Rahman  
Email: [mdsaifur.rahman.1@und.edu](mailto:mdsaifur.rahman.1@und.edu)  
Date: 03/19/2021

```
fig = plt.gcf()
```

```
# Set size of figure to fit 4x4 configuration pics
```

```
fig.set_size_inches(ncols * 4, nrows * 4)
```

```
pic_index += 8
```

```
next_cat_pix = [os.path.join(train_cats_dir, fname)
```

```
    for fname in train_cat_fnames[pic_index-8:pic_index]]
```

```
next_dog_pix = [os.path.join(train_dogs_dir, fname)
```

```
    for fname in train_dog_fnames[pic_index-8:pic_index]]
```

```
for i, img_path in enumerate(next_cat_pix+next_dog_pix):
```

```
    # Set up subplot; subplot indices start at 1
```

```
    sp = plt.subplot(nrows, ncols, i + 1)
```

```
    sp.axis('Off') # Don't show axes (or gridlines)
```

```
    img = mpimg.imread(img_path)
```

```
    plt.imshow(img)
```

```
plt.show()
```

```
# Building a Small Convnet from Scratch
```

```
# Our input feature map is 150x150x3: 150x150 for the image pixels, and 3 for the three color  
channels: R, G, and B
```

```
img_input = layers.Input(shape=(150, 150, 3))
```

```
# First convolution extracts 16 filters that are 3x3
```

```
# Convolution is followed by max-pooling layer with a 2x2 window
```

Submitted by: Md. Saifur Rahman  
Email: [mdsaifur.rahman.1@und.edu](mailto:mdsaifur.rahman.1@und.edu)  
Date: 03/19/2021

# Relu overcome vanishing gradient problem, learning faster and better. Relu is default for Convolutional Neural Network.

```
x = layers.Conv2D(16, 3, activation='relu')(img_input)
```

```
x = layers.MaxPooling2D(2)(x)
```

# Second convolution extracts 32 filters that are 3x3

# Convolution is followed by max-pooling layer with a 2x2 window

```
x = layers.Conv2D(32, 3, activation='relu')(x)
```

```
x = layers.MaxPooling2D(2)(x)
```

# Third convolution extracts 64 filters that are 3x3

# Convolution is followed by max-pooling layer with a 2x2 window

```
x = layers.Conv2D(64, 3, activation='relu')(x)
```

```
x = layers.MaxPooling2D(2)(x)
```

# Flatten feature map to a 1-dim tensor so we can add fully connected layers

```
x = layers.Flatten()(x)
```

# Create a fully connected layer with ReLU activation and 512 hidden units

```
x = layers.Dense(512, activation='relu')(x)
```

# Create output layer with a single node and sigmoid activation

```
output = layers.Dense(1, activation='sigmoid')(x)
```

# Create model:

# input = input feature map

# output = input feature map + stacked convolution/maxpooling layers + fully

# connected layer + sigmoid output layer

```
model = Model(img_input, output)
```

# summarize the model architecture

# The convolution layers reduce the size of the feature maps by a bit due to padding, and each pooling layer halves the feature map.

model.summary()

# We will train our model with the binary\_crossentropy loss, because it's a binary classification problem and our final activation is a sigmoid.

# We will use the rmsprop optimizer with a learning rate of 0.001 (Other optimizers, such as Adam and Adagrad, also automatically adapt the learning rate during training, and would work equally well here.)

```
model.compile(loss='binary_crossentropy',  
              optimizer=RMSprop(lr=0.001),  
              metrics=['acc'])
```

# Data Preprocessing

# Let's set up data generators that will read pictures in our source folders, convert them to float32 tensors, and feed them (with their labels) to our network.

# We'll have one generator for the training images and one for the validation images.

# Our generators will yield batches of 20 images of size 150x150 and their labels (binary).

# we will preprocess our images by normalizing the pixel values to be in the [0, 1] range (originally all values are in the [0, 255] range).

# All images will be rescaled by 1./255 (min-max normalization)

# In Keras this can be done via the keras.preprocessing.image.ImageDataGenerator class using the rescale parameter.

```
train_datagen = ImageDataGenerator(rescale=1./255)
```

```
val_datagen = ImageDataGenerator(rescale=1./255)
```

Submitted by: Md. Saifur Rahman  
Email: [mdsaifur.rahman.1@und.edu](mailto:mdsaifur.rahman.1@und.edu)  
Date: 03/19/2021

# Flow training images in batches of 20 using train\_datagen generator

```
train_generator = train_datagen.flow_from_directory(  
    train_dir, # This is the source directory for training images  
    target_size=(150, 150), # All images will be resized to 150x150  
    batch_size=20,  
    # Since we use binary_crossentropy loss, we need binary labels  
    class_mode='binary')
```

# Flow validation images in batches of 20 using val\_datagen generator

```
validation_generator = val_datagen.flow_from_directory(  
    validation_dir,  
    target_size=(150, 150),  
    batch_size=20,  
    class_mode='binary')
```

# Training

```
history = model.fit(  
    train_generator,  
    steps_per_epoch=100, # 2000 images = batch_size * steps  
    epochs=15,  
    validation_data=validation_generator,  
    validation_steps=50, # 1000 images = batch_size * steps  
    verbose=2)
```

# Visualizing Intermediate Representations

# we go from the raw pixels of the images to increasingly abstract and compact representations.

# Let's define a new Model that will take an image as input, and will output intermediate representations for all layers in the previous model after the first.

Submitted by: Md. Saifur Rahman  
Email: [mdsaifur.rahman.1@und.edu](mailto:mdsaifur.rahman.1@und.edu)  
Date: 03/19/2021

# These representations carry increasingly less information about the original pixels of the image, but increasingly refined information about the class of the image.

```
successive_outputs = [layer.output for layer in model.layers[1:]]
```

```
visualization_model = Model(img_input, successive_outputs)
```

```
# Let's prepare a random input image of a cat or dog from the training set.
```

```
cat_img_files = [os.path.join(train_cats_dir, f) for f in train_cat_fnames]
```

```
dog_img_files = [os.path.join(train_dogs_dir, f) for f in train_dog_fnames]
```

```
img_path = random.choice(cat_img_files + dog_img_files)
```

```
img = load_img(img_path, target_size=(150, 150)) # this is a PIL image
```

```
x = img_to_array(img) # Numpy array with shape (150, 150, 3)
```

```
x = x.reshape((1,) + x.shape) # Numpy array with shape (1, 150, 150, 3)
```

```
# Rescale by 1/255
```

```
x /= 255
```

```
# Let's run our image through our network, thus obtaining all
```

```
# intermediate representations for this image.
```

```
successive_feature_maps = visualization_model.predict(x)
```

```
# These are the names of the layers, so can have them as part of our plot
```

```
layer_names = [layer.name for layer in model.layers]
```

```
# Now let's display our representations
```

```
for layer_name, feature_map in zip(layer_names, successive_feature_maps):
```

```
    if len(feature_map.shape) == 4:
```

```
        # Just do this for the conv / maxpool layers, not the fully-connected layers
```

Submitted by: Md. Saifur Rahman  
Email: [mdsaifur.rahman.1@und.edu](mailto:mdsaifur.rahman.1@und.edu)  
Date: 03/19/2021

```
n_features = feature_map.shape[-1] # number of features in feature map

# The feature map has shape (1, size, size, n_features)

size = feature_map.shape[1]

# We will tile our images in this matrix

display_grid = np.zeros((size, size * n_features))

for i in range(n_features):

    # Postprocess the feature to make it visually palatable

    x = feature_map[0, :, :, i]

    x -= x.mean()

    x /= x.std()

    x *= 64

    x += 128

    x = np.clip(x, 0, 255).astype('uint8')

    # We'll tile each filter into this big horizontal grid

    display_grid[:, i * size : (i + 1) * size] = x

# Display the grid

scale = 20. / n_features

plt.figure(figsize=(scale * n_features, scale))

plt.title(layer_name)

plt.grid(False)

plt.imshow(display_grid, aspect='auto', cmap='viridis')


# Evaluating Accuracy and Loss for the Model


# Retrieve a list of accuracy results on training and validation data

# sets for each training epoch

acc = history.history['acc']

val_acc = history.history['val_acc']
```

```
# Retrieve a list of list results on training and validation data
```

```
# sets for each training epoch
```

```
loss = history.history['loss']
```

```
val_loss = history.history['val_loss']
```

```
# Get number of epochs
```

```
epochs = range(len(acc))
```

```
# Plot training and validation accuracy per epoch
```

```
a, = plt.plot(epochs, acc)
```

```
b, = plt.plot(epochs, val_acc)
```

```
plt.title("Training and validation accuracy")
```

```
plt.legend([a, b],["acc", "val_acc"])
```

```
plt.show()
```

```
# Plot training and validation loss per epoch
```

```
c,= plt.plot(epochs, loss)
```

```
d,= plt.plot(epochs, val_loss)
```

```
plt.title("Training and validation loss")
```

```
plt.legend([c, d],["loss", "val_loss"])
```

```
print("Training Data Accuracy:",max(acc))
```

```
print("Training Data Loss:",max(loss))
```

```
print("Validation Data Accuracy",max(val_acc))
```

```
print("Validation Data Loss",max(val_loss))
```

```
# terminate the kernel and free memory resources
```

Submitted by: Md. Saifur Rahman  
Email: [mdsaifur.rahman.1@und.edu](mailto:mdsaifur.rahman.1@und.edu)  
Date: 03/19/2021

```
#import os, signal
```

```
#os.kill(os.getpid(), signal.SIGKILL)
```