

ECEN 4493 Homework 4

Keenan Holsapple

April 12 2023

Contents

1	Introduction	2
1.1	Problems Defined	2
2	Part 1	3
2.1	Outputs	3
2.2	Code	5
3	Part 2	7
3.1	Designing the CNN	7
3.2	Outputs	8
3.3	Code	9
4	Part 3	11
4.1	Output	11
4.2	Code	13
5	Academic Statement	17
6	References	18

1 Introduction

The goal of this homework assignment is to learn how to implement a deep neural network (DNN) and convolutional neural network (CNN). To practice the implementation, we utilize existing libraries of data that can apply to the desired outcome. Each part in this assignment uses a different library that exists and can be split properly with an associated library. All codes in this assignment are written in Python and graphs are displayed utilizing the Matplotlib library that can be installed.

1.1 Problems Defined

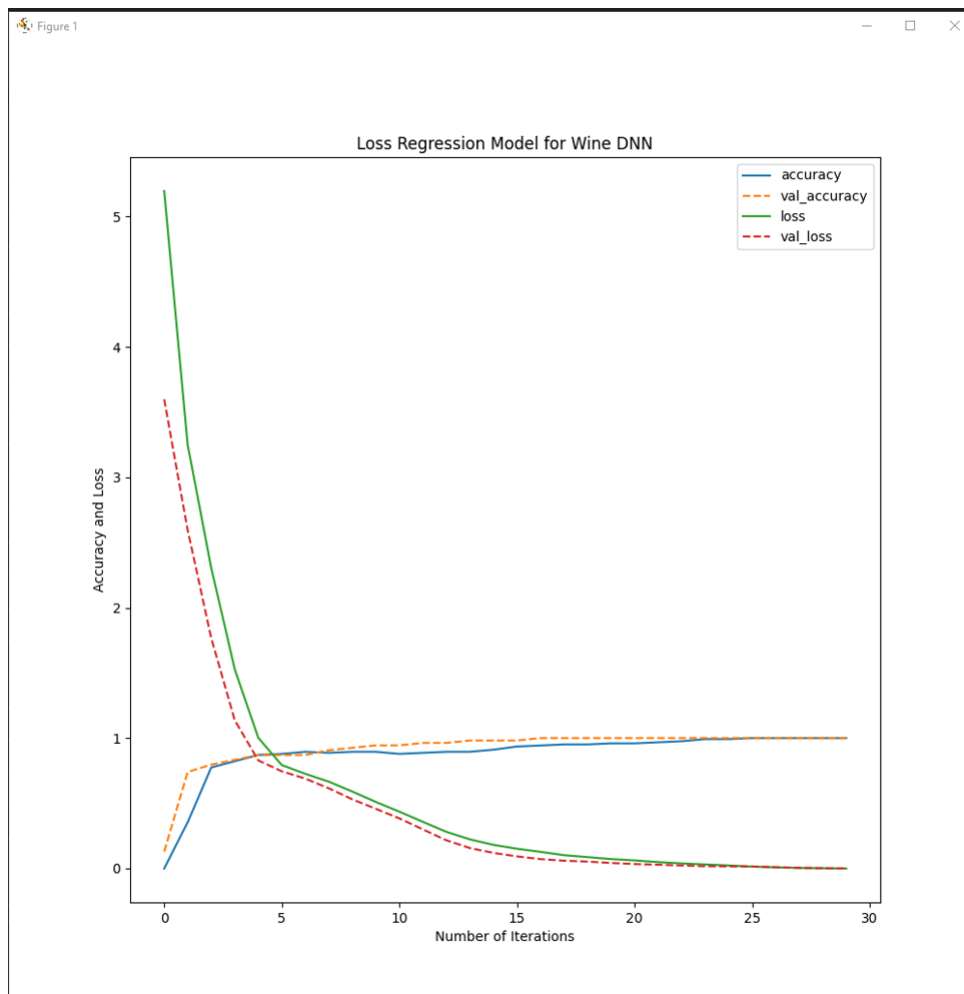
The assignment's parts are split up as:

- Part 1: Build a neural network to predict wine class using the “Wine” dataset as used in previous homework. Use the “Adam” optimizer. Choice of neural network is yours. Plot the “Accuracy vs. Iterations” and “Loss (Both Training and Validation Loss) vs. Iterations”. Also, based on the performance on the test set, write your accuracy score.
- Part 2: Build a CNN to predict the MNIST dataset. Choice of layers is your choice. Use the Adam optimizer and plot the same charts as the first question. Also, write your accuracy score. Write a brief paragraph regarding your choice of layers.
- Part 3: Build a CNN to classify the “CIFAR10” dataset. Then load the pretrained “ResNet50” model and remove the last layer. Add additional layers based on the dataset. Compare the accuracy results of your model with the modified pre-trained model.

2 Part 1

2.1 Outputs

This was a very well-tuned output for determining wines. I found that compounding the number of neurons in the three layers outside of the input layer worked best. I fixed the model to execute at 30 epochs- however, the model had a tendency to overlearn. To prevent this, I included an early stop function included in the TensorFlow library that allowed it to stop at an acceptable time.



```
PROBLEMS 33 OUTPUT TERMINAL DEBUG CONSOLE COMMENTS

Epoch 8/30
4/4 [=====] - 0s 9ms/step - loss: 0.6660
Epoch 9/30
4/4 [=====] - 0s 9ms/step - loss: 0.5898
Epoch 10/30
4/4 [=====] - 0s 9ms/step - loss: 0.5104
Epoch 11/30
4/4 [=====] - 0s 8ms/step - loss: 0.4360
Epoch 12/30
4/4 [=====] - 0s 9ms/step - loss: 0.3576
Epoch 13/30
4/4 [=====] - 0s 9ms/step - loss: 0.2811
Epoch 14/30
4/4 [=====] - 0s 8ms/step - loss: 0.2236
Epoch 15/30
4/4 [=====] - 0s 8ms/step - loss: 0.1813
Epoch 16/30
4/4 [=====] - 0s 8ms/step - loss: 0.1515
Epoch 17/30
4/4 [=====] - 0s 8ms/step - loss: 0.1275
Epoch 18/30
4/4 [=====] - 0s 9ms/step - loss: 0.1020
Epoch 19/30
4/4 [=====] - 0s 9ms/step - loss: 0.0873
Epoch 20/30
4/4 [=====] - 0s 9ms/step - loss: 0.0726
Epoch 21/30
4/4 [=====] - 0s 9ms/step - loss: 0.0623
Epoch 22/30
4/4 [=====] - 0s 9ms/step - loss: 0.0493
Epoch 23/30
4/4 [=====] - 0s 9ms/step - loss: 0.0391
Epoch 24/30
4/4 [=====] - 0s 9ms/step - loss: 0.0312
Epoch 25/30
4/4 [=====] - 0s 9ms/step - loss: 0.0235
Epoch 26/30
4/4 [=====] - 0s 8ms/step - loss: 0.0148
Epoch 27/30
4/4 [=====] - 0s 9ms/step - loss: 0.0091
Epoch 28/30
4/4 [=====] - 0s 9ms/step - loss: 0.0043
Epoch 29/30
4/4 [=====] - 0s 9ms/step - loss: 0.0026
Epoch 30/30
4/4 [=====] - 0s 9ms/step - loss: 1.5843
2/2 [=====] - 0s 2ms/step - loss: 6.1881
Test Loss is 0.0006188057013787329
Test Accuracy is 1.0
[]
```

2.2 Code

```
# ECEN4493 HW4 Part 1
# Keenan Holsapple
# DNN for Wine library
# For libraries, "pip install -r requirements.txt"
# To execute, "python part1_keenan_holsapple.py"

# Outputs: procedural epoch information, loss/accuracy values,
# loss+accuracy vs iterations graph

import matplotlib.pyplot as plt
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from keras.models import Sequential
from keras.layers import Dense
from tensorflow.keras.callbacks import EarlyStopping

# loading in dataset from website
vine = load_wine()

# defining x2_train to hold all data
X2_train = vine.data

# defining x2_test to hold all test values
X2_test = vine.target.astype('int')

# splitting data into training values
X_train, X_test, y_train, y_test = train_test_split(X2_train,
                                                    X2_test, test_size=0.3,
                                                    random_state=0)

# scaling data using existing library; keeps learning consistent
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# stops model from overlearning
early_stop = EarlyStopping(monitor='val_loss', patience=5,
                           restore_best_weights=True)

# defining neural network with layers and neurons
model = Sequential([
    # input layer
    Dense(units=(X_train.shape[1]), input_shape=(X_train.shape[1],),
          activation='relu'),
    # layer 1
    Dense(units=64, activation='relu'),
    # layer 2
    Dense(units=128, activation='relu'),
    # layer 3
    Dense(units=256, activation='relu')
])

# compiler defining adam as the optimiser
```

```

# loss needs to be sparse_categorical_crossentropy since out since
# test values aren't binary,
# measures accuracy
model.compile(optimizer='adam', loss='
    sparse_categorical_crossentropy',
    metrics=['accuracy'])

# training the model with data at 30 epochs fixed, incoming test
# data form wine list, and early
# stop defined
train = model.fit(X_train, y_train, validation_data=(X_test,y_test)
    , epochs=30, callbacks=[
    early_stop])

# computing values for loss and accuracy in the model
loss, accuracy = model.evaluate(X_test, y_test)

# defining plot figure
fig, ax = plt.subplots(figsize=(10,10))

# labeling plot
plt.plot(train.history['accuracy'], label='accuracy')
# measuring accuracy of accuracy
plt.plot(train.history['val_accuracy'], label='val_accuracy',
    linestyle='--')
plt.plot(train.history['loss'], label='loss')
# measuing accuracy of loss
plt.plot(train.history['val_loss'], label='val_loss', linestyle='--
    ')

plt.legend()
plt.xlabel("Number of Iterations")
plt.ylabel("Accuracy and Loss")
plt.title("Loss Regression Model for Wine DNN")

# output to line value of loss and accuracy
print("Loss Value: " + str(loss))
print("Accuracy Value: " + str(accuracy))

# show plot
plt.show()

```

3 Part 2

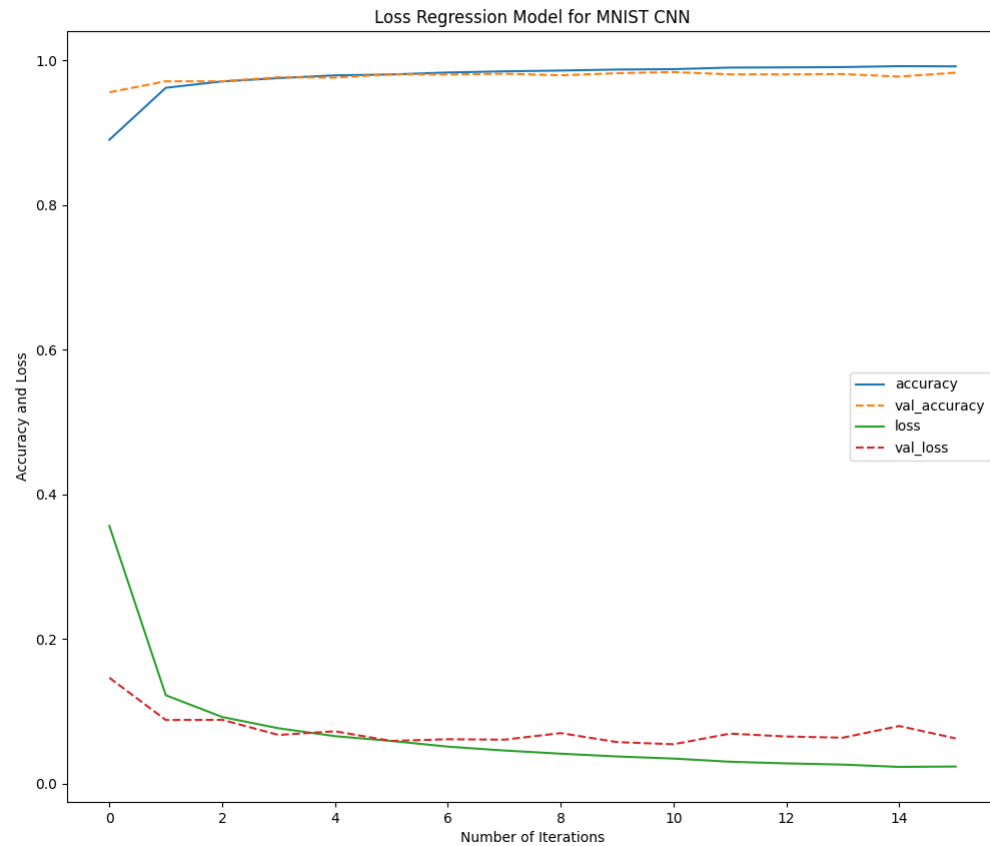
3.1 Designing the CNN

The model I designed for the MNIST dataset is:

- Two convolutional layers, 4 filters and a 2x2 filter size.
- Two pooling layers at 2x2 size
- A flatten layer to read the data
- Two dense layers for analyzing, 64, 128, respectively. I've found that compounding values helps performance in this model.
- A final dense layer for output with 10 neurons.

I predetermined having more than one convolving layer to verify the previous convolution's accuracy. After experimenting, it seemed unnecessary to have any more than two layers. Since I had two convolutional layers, I also needed 2 pooling layers with a corresponding window size. The flatten is to reduce the data to a 1 dimensional layer, which allows the measurements to be determined in the upcoming dense layers. I determined that having dense layers was necessary since the model starts at such a high accuracy, allowing it to make those fine-tuning decisions in the later epochs. I studied the number of neurons in these two dense layers and found that 64 and 128 were sufficient.

3.2 Outputs



```
PS C:\Users\19188\Desktop\spring_2023\ai\hw4\ECEN4493_AI_Projects> python part2_keenan_holsapple.py
Epoch 1/3
1875/1875 [=====] - 28s 15ms/step - loss: 0.1320 - accuracy: 0.9593 - val_loss: 0.0439 - val_accuracy: 0.9889
Epoch 2/3
1875/1875 [=====] - 27s 14ms/step - loss: 0.0429 - accuracy: 0.9872 - val_loss: 0.0347 - val_accuracy: 0.9889
Epoch 3/3
1875/1875 [=====] - 27s 14ms/step - loss: 0.0314 - accuracy: 0.9900 - val_loss: 0.0341 - val_accuracy: 0.9889
313/313 [=====] - 1s 3ms/step - loss: 0.0341 - accuracy: 0.9889
Loss Value: 0.03409941866993904
Accuracy Value: 0.9889000058174133
[]
```


3.3 Code

```
# ECEN4493 HW4 Part 2
# Keenan Holsapple
# DNN for Wine library
# For libraries, "pip install -r requirements.txt"
# To execute, "python part2_keenan_holsapple.py"

# Outputs: procedural epoch information, loss/accuracy values,
# loss+accuracy vs iterations graph

import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation,
                                Flatten, Conv2D, MaxPooling2D
from tensorflow.keras.callbacks import EarlyStopping

# splitting incoming data
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Normalize pixel values to be between 0 and 1
X_train = X_train / 255.0
X_test = X_test / 255.0

# defining early stopping
early_stop = EarlyStopping(monitor='val_loss', patience=5,
                           restore_best_weights=True)

# defining dnn model at 7 layers
model = Sequential()

# input convolutional layer 1 (1)
model.add(Conv2D(4, (2,2), input_shape=(28, 28, 1)))
model.add(Activation("relu"))

# pooling layer 1 (2)
model.add(MaxPooling2D(pool_size=(2,2)))

# convolutional layer 2 (3)
model.add(Conv2D(4, (2,2)))
model.add(Activation("relu"))

# pooling layer 2 (4)
model.add(MaxPooling2D(pool_size=(2,2)))

# flattening layer 1 (5)
model.add(Flatten())

# dense layer 1 (6)
model.add(Dense(64, activation="relu"))

model.add(Dense(128, activation="relu"))
```

```

# dense layer 2 (7)
model.add(Dense(10,
                # using softmax since 0-9 output measurements
                activation='softmax'))

# loss measured with multiple different output variables, adam
# optimizer, and measuring for
# accuracy in model
model.compile(loss="sparse_categorical_crossentropy", optimizer="
              adam", metrics=['accuracy'])

# training specifications
train = model.fit(X_train.reshape(-1, 28, 28, 1), y_train,

                  # reshaping: size input data, height, width,
                  # channel
                  validation_data=(X_test.reshape(-1, 28, 28, 1),
                                   y_test),

                  # number of epochs
                  epochs=20,

                  # stops overlearning in case
                  callbacks=[early_stop])

# finding loss and accuracy depending on inputs
loss, accuracy = model.evaluate(X_test.reshape(-1, 28, 28, 1),
                                y_test)

model.save('mnist_cnn.h5')

# plot specifications
fig, ax = plt.subplots(figsize=(12,10))
plt.plot(train.history['accuracy'], label='accuracy')
plt.plot(train.history['val_accuracy'], label='val_accuracy',
         linestyle='--')
plt.plot(train.history['loss'], label='loss')
plt.plot(train.history['val_loss'], label='val_loss', linestyle='--')
plt.legend()
plt.xlabel("Number of Iterations")
plt.ylabel("Accuracy and Loss")
plt.title("Loss Regression Model for MNIST CNN")

# output to line value of loss and accuracy
print("Loss Value: " + str(loss))
print("Accuracy Value: " + str(accuracy))

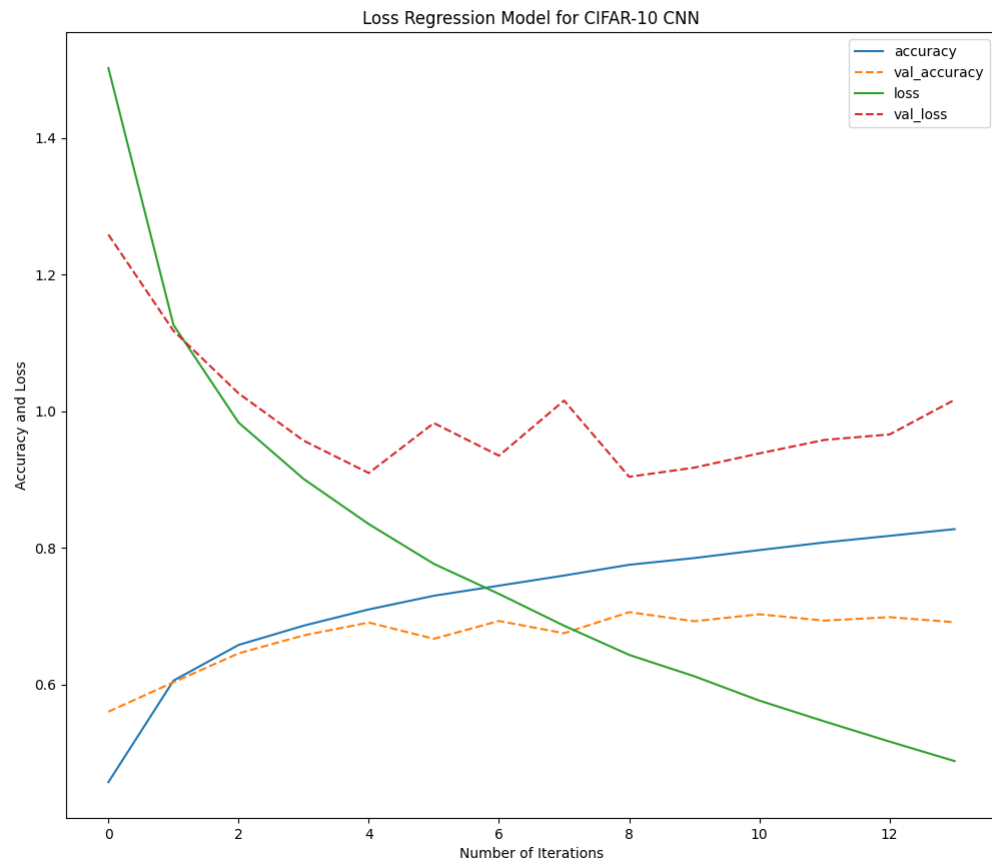
# show plot
plt.show()

```

4 Part 3

4.1 Output

First, here are the training results using CIFAR10 on the CNN I designed:



The loss/accuracy values at the end of the final epoch is listed here:

```

Epoch 1/15
1563/1563 [=====] - 58s 36ms/step - loss: 1.5025 - accuracy: 0.4573 - val_loss: 1.2587 - val_accuracy: 0.4573
Epoch 2/15
1563/1563 [=====] - 58s 37ms/step - loss: 1.1264 - accuracy: 0.6059 - val_loss: 1.1179 - val_accuracy: 0.6059
Epoch 3/15
1563/1563 [=====] - 58s 37ms/step - loss: 0.9836 - accuracy: 0.6581 - val_loss: 1.0264 - val_accuracy: 0.6581
Epoch 4/15
1563/1563 [=====] - 58s 37ms/step - loss: 0.9010 - accuracy: 0.6863 - val_loss: 0.9572 - val_accuracy: 0.6863
Epoch 5/15
1563/1563 [=====] - 59s 38ms/step - loss: 0.8350 - accuracy: 0.7101 - val_loss: 0.9097 - val_accuracy: 0.7101
Epoch 6/15
1563/1563 [=====] - 58s 37ms/step - loss: 0.7768 - accuracy: 0.7301 - val_loss: 0.9828 - val_accuracy: 0.7301
Epoch 7/15
1563/1563 [=====] - 57s 37ms/step - loss: 0.7328 - accuracy: 0.7449 - val_loss: 0.9351 - val_accuracy: 0.7449
Epoch 8/15
1563/1563 [=====] - 57s 37ms/step - loss: 0.6862 - accuracy: 0.7597 - val_loss: 1.0158 - val_accuracy: 0.7597
Epoch 9/15
1563/1563 [=====] - 57s 37ms/step - loss: 0.6435 - accuracy: 0.7754 - val_loss: 0.9042 - val_accuracy: 0.7754
Epoch 10/15
1563/1563 [=====] - 57s 37ms/step - loss: 0.6123 - accuracy: 0.7853 - val_loss: 0.9176 - val_accuracy: 0.7853
Epoch 11/15
1563/1563 [=====] - 59s 38ms/step - loss: 0.5766 - accuracy: 0.7970 - val_loss: 0.9385 - val_accuracy: 0.7970
Epoch 12/15
1563/1563 [=====] - 60s 38ms/step - loss: 0.5462 - accuracy: 0.8081 - val_loss: 0.9582 - val_accuracy: 0.8081
Epoch 13/15
1563/1563 [=====] - 57s 37ms/step - loss: 0.5167 - accuracy: 0.8178 - val_loss: 0.9662 - val_accuracy: 0.8178
Epoch 14/15
1563/1563 [=====] - 58s 37ms/step - loss: 0.4881 - accuracy: 0.8277 - val_loss: 1.0169 - val_accuracy: 0.8277
313/313 [=====] - 3s 9ms/step - loss: 0.9042 - accuracy: 0.7061
Loss Value: 0.9041997790336609
Accuracy Value: 0.7060999870300293

```

To compare the accuracy, I tested the RESNET50 model and CIFAR10 model I developed with a few pictures. After saving my model, I could quickly access the CIFAR10 trained model and loop several pictures to see outputs.

The RESNET50 is much more accurate than the CIFAR10. It is not only more tuned in hyperparameters, but it also has more data to go off of compared to CIFAR10. It also uses more data, leaving it to be able to conclude more about a picture.

For instance, here are two pictures that show the accuracy of the RESNET50 and innaccuracy of the CIFAR10:



CIFAR10 determined these two pictures to be:

```
keyboardInterrupt
PS C:\Users\19188\Desktop\spring_2023\ai\hw4\ECEN4493_AI_Projects> python part3_cifar10_test.py
1/1 [=====] - 0s 87ms/step
Predicted: [[1. 0. 0. 0. 0. 0. 0. 0.]]
Predicted label: airplane
```

```
PS C:\Users\19188\Desktop\spring_2023\ai\hw4\ECEN4493_AI_Projects> python test.py
1/1 [=====] - 0s 88ms/step
Predicted: [[0. 1. 0. 0. 0. 0. 0. 0.]]
Predicted label: automobile
```

With the model obviously not being able to determine the bird properly but guessing the car.

The RESNET50 model's determinations were:

```
PS C:\Users\19188\Desktop\spring_2023\ai\hw4\ECEN4493_AI_Projects> python part3_resnet_test.py
1/1 [=====] - 1s 839ms/step
Predicted: [('n02690373', 'airliner', 0.8400545), ('n04552348', 'warplane', 0.14375448), ('n04592741', 'wing', 0.0125857815)]
PS C:\Users\19188\Desktop\spring_2023\ai\hw4\ECEN4493_AI_Projects> python part3_resnet_test.py
1/1 [=====] - 1s 822ms/step
Predicted: [('n04266014', 'space shuttle', 0.98286384), ('n02951585', 'can opener', 0.0063388837), ('n03109150', 'corkscrew', 0.00154515)]
PS C:\Users\19188\Desktop\spring_2023\ai\hw4\ECEN4493_AI_Projects> python part3_resnet_test.py
1/1 [=====] - 1s 815ms/step
Predicted: [('n01534433', 'junco', 0.2064988), ('n01531178', 'goldfinch', 0.2020626), ('n01560419', 'bulbul', 0.13693014)]
PS C:\Users\19188\Desktop\spring_2023\ai\hw4\ECEN4493_AI_Projects> 
```

```
PS C:\Users\19188\Desktop\spring_2023\ai\hw4\ECEN4493_AI_Projects> python part3_resnet_test.py
1/1 [=====] - 1s 988ms/step
Predicted: [('n04285008', 'sports car', 0.9849406), ('n04037443', 'racer', 0.0065542455), ('n03100240', 'convertible', 0.003879786)]
PS C:\Users\19188\Desktop\spring_2023\ai\hw4\ECEN4493_AI_Projects> 
```

The RESNET50 model was able to not only determine almost all of the pictures accurately, but also was able to determine even more detailed aspects of a picture.

4.2 Code

CIFAR10 Model:

```
# ECEN4493 HW4 Part 3
# Keenan Holsapple
# CNN for CIFAR10 database
# For libraries, "pip install -r requirements.txt"
# To execute, "python part3_keenán_holsapple.py"

# Outputs: procedural epoch information, loss/accuracy values,
# loss+accuracy vs iterations graph

import tensorflow as tf
import matplotlib.pyplot as plt
```

```

from keras.datasets import cifar10

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation,
                                Flatten, Conv2D, MaxPooling2D
from tensorflow.keras.callbacks import EarlyStopping

import ssl

ssl._create_default_https_context = ssl._create_unverified_context

# splitting incoming data
(X_train, y_train), (X_test, y_test) = cifar10.load_data()

# Normalize pixel values to be between 0 and 1
X_train = X_train / 255.0
X_test = X_test / 255.0

# defining early stopping
early_stop = EarlyStopping(monitor='val_loss', patience=5,
                           restore_best_weights=True)

# defining dnn model at 7 layers
model = Sequential()

# input convolutional layer 1
model.add(Conv2D(128, (3,3), input_shape=(32, 32, 3)))
model.add(Activation("relu"))

# pooling layer 1
model.add(MaxPooling2D(pool_size=(2,2)))

# convolutional layer 2
model.add(Conv2D(128, (3,3)))
model.add(Activation("relu"))

# pooling layer 2
model.add(MaxPooling2D(pool_size=(2,2)))

# flattening layer 1
model.add(Flatten())

# dense layer 1
model.add(Dense(64, activation="relu"))

# dense layer 2
model.add(Dense(10, activation='softmax'))

# loss measured with multiple different output variables, adam
# optimizer, and measuring for accuracy in model
model.compile(loss="sparse_categorical_crossentropy", optimizer="
adam", metrics=['accuracy'])

# training specifications

```

```

train = model.fit(X_train, y_train,
                  validation_data=(X_test, y_test),
                  epochs=15,
                  callbacks=[early_stop])

# finding loss and accuracy depending on inputs
loss, accuracy = model.evaluate(X_test, y_test)

model.save('cifar10_cnn.h5')

# plot specifications
fig, ax = plt.subplots(figsize=(12,10))
plt.plot(train.history['accuracy'], label='accuracy')
plt.plot(train.history['val_accuracy'], label='val_accuracy',
         linestyle='--')
plt.plot(train.history['loss'], label='loss')
plt.plot(train.history['val_loss'], label='val_loss', linestyle='--')

plt.legend()
plt.xlabel("Number of Iterations")
plt.ylabel("Accuracy and Loss")
plt.title("Loss Regression Model for CIFAR-10 CNN")

# output to line value of loss and accuracy
print("Loss Value: " + str(loss))
print("Accuracy Value: " + str(accuracy))

# show plot
plt.show()

```

CIFAR10 Test:

```

import tensorflow as tf
import numpy as np
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import img_to_array,
load_img
from tensorflow.keras.applications.resnet50 import preprocess_input
, decode_predictions

# Load the CIFAR-10 CNN model
model_path = 'cifar10_cnn.h5'
model = load_model(model_path)

# Load and preprocess the test image
img_path = 'bird_1.jpg'
img = load_img(img_path, target_size=(32, 32))
x = img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

# Make a prediction
preds = model.predict(x)
print('Predicted:', preds)

# Decode the prediction result

```

```

class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']
pred_class = np.argmax(preds)
pred_label = class_names[pred_class]
print('Predicted label:', pred_label)

```

RESNET50 Test:

```

from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.resnet50 import preprocess_input
                                     , decode_predictions
import numpy as np

model = ResNet50(weights='imagenet')

# test image(can implement loop for reading multitude as well)
img_path = 'test_images/bird_1.jpg'
# adapting image to test
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

# prediction
preds = model.predict(x)

# decoding prediction with embedded function in resnet50
print('Predicted:', decode_predictions(preds, top=3)[0])


```


5 Academic Statement

Statement of Academic Honesty:

For this homework, I make the following truthful statements:

- I have not received, I have not given, nor will I give or receive, any assistance to another student taking this quiz.
- I will not plagiarize someone else's work and turn it in as my own.
- I understand that acts of academic dishonesty may be penalized to the full extent allowed by the University Student Conduct Code, including receiving a failing grade (F!) for the course. I recognize that I am responsible for understanding the provisions of the University Student Conduct Code as they relate to this academic exercise.


Your Signature

6 References

- TensorFlow DNN library with Keras: <https://machinelearningmastery.com/tensorflow-tutorial-deep-learning-with-tf-keras/>
- Writing a CNN with TensorFlow: https://www.youtube.com/watch?v=WvoLTXIjBYU&ab_channel=sentdex
- RESNET50 implementation:
 - Main document: <https://keras.io/api/applications/resnet/#resnet50-function>
 - Python implementation: <https://medium.com/@ashabb/image-recognition-with-model-resnet>
- Tuning Parameters: https://www.youtube.com/watch?v=6Nf1x7qThR8&t=533s&ab_channel=GregHogg