Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

CS7CS4/CSU44061 - Machine Learning
Assignment 6

Student Name
Hugh Jordan

Student Number
16321743

Assignment 6

(i) Convolutions

(a) A function was implemented that takes an $n \times n$ array, and a $k \times k$ kernel as the inputs, convolves the input array using the kernel and then returns the result. The code used to implement this function is shown in Figure 1. The program begins by asserting that the dimensions of the array and kernel are both of the form $n \times n$ and $k \times k$. The program then flips the kernel and initialises an output array of zeroes. The equation to calculate a single dimension of the output array was derived using the following equation,

$$D = \frac{(W + 2P - K)}{S} + 1,$$

where W is the dimension of the array, P is the dimension of the padding, K is the dimension of the kernel, and S is the stride. However, the equation could be simplified to $D = W - K + 1$ because no padding or stride was applied to the input array. Finally, the output array was filled up by looping through each possible subarray in the input array and convolving the subarray with the kernel.

```
6  def conv2d(array, kernel):
7      n, n0 = array.shape
8      assert n == n0
9      k, k0 = kernel.shape
10     assert k == k0
11
12     kernel = np.flipud(np.fliplr(kernel))
13     output = np.zeros((n - k + 1, n - k + 1))
14
15     for y in range(output.shape[1]):
16         for x in range(output.shape[0]):
17             output[y, x] = (kernel * array[y:y+k, x:x+k]).sum()
18     return output
```

Figure 1. 2D-Convolution Implementation

(b) The selected image is shown in Figure 2. The image was convolved using the two provided kernels, and the resultant images are shown in Figures 3 and 4, respectively. It is evident from Figure 3 and 4 that both kernels are variations of edge detection filters. This was concluded because the edges of the icon are highlighted in Figure 3 and 4.

Figure 2. Original Image

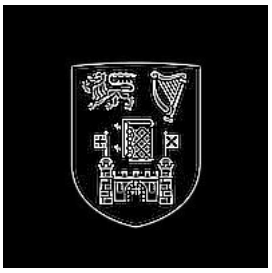

Figure 3. Convolved using kernel1

Figure 4. Convolved using kernel2

(ii) Convolutional Neural Network

(a) The architecture of the model is shown below in Figure 5. It is evident from Figure 5 that the architecture consists of four convolutional layers followed by a single dense layer. The first convolutional layer has 16 output filters, a kernel of size (3, 3), zero-padding and a relu activation function. Similarly, the second convolutional layer has 16 output filters, a kernel of size (3, 3), zero-padding and a relu activation function as well. However, this layer also has a stride of (2, 2). The stride is the number of pixels the kernel moves between convolutions.

The third convolutional layer has 32 output filters, a kernel of size (3, 3), zero-padding and a relu activation function. Similarly, the fourth convolutional layer has 32 output filters, a kernel of size (3, 3), zero-padding and a relu activation function as well. Additionally, this layer has a stride of (2,2). Dropout was then applied to the fourth convolutional layer. Dropout is a type of regularisation that turns off a number of the nodes during training to improve their ability to generalise. The number of nodes turned off is set by assigning a value between 0 and 1 to the parameter of the dropout function. Finally, a dense layer with a softmax activation function was used to output the predictions of the model, where the dimensions of the dense layer are equal to the number of classes of the data.

```
33    model = keras.Sequential()
34    model.add(Conv2D(16, (3,3), padding='same', input_shape=x_train.shape[1:],activation='relu'))
35    model.add(Conv2D(16, (3,3), strides=(2,2), padding='same', activation='relu'))
36    model.add(Conv2D(32, (3,3), padding='same', activation='relu'))
37    model.add(Conv2D(32, (3,3), strides=(2,2), padding='same', activation='relu'))
38    model.add(Dropout(0.5))
39    model.add(Flatten())
40    model.add(Dense(num_classes, activation='softmax',kernel_regularizer=regularizers.l1(0.0001)))
41    model.compile(loss="categorical_crossentropy", optimizer='adam', metrics=["accuracy"])
```

Figure 5. ConvNet Architecture

(b) (i) The Keras model has 37,146 parameters. This value was obtained using the Keras function summary. The performance of the model is shown in Figure 6. It is evident from Figure 6 that the model performed worse on the validation data when compared with the training data. This is more than likely due to overfitting. The performance of the model was also compared against the performance of a baseline model which always predicts the most common label. This baseline model was created using the sklearn function Dummy Classifier. This function required the input array y to be flattened in order to train and validate the model.

(ii) It can be deduced from Figure 6 that the model suffers from overfitting. Overfitting is when the model performs better during training than during validation. This is because the model begins to fit the noise in the training data and generalises poorly outside of the training data. Dropout is one type of regularisation that is used to reduce overfitting. In fact, dropout used in the above architecture. However, the effect of increasing the size of the training data and L_1 regularisation has on overfitting will be explored as well.

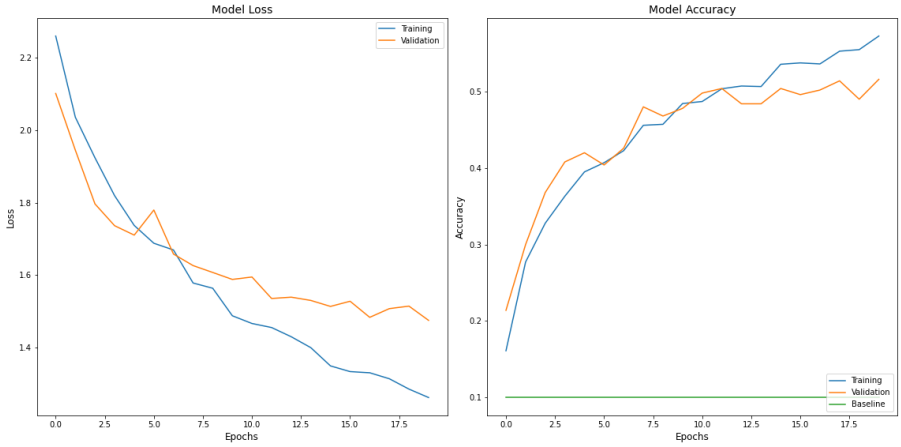

Figure 6. Performance of Model when training data size = 5000

(iii) The effect of increasing the amount of training data had on the performance of the model was explored. This was performed by training the model using the range of training data sizes [5000, 10000, 20000, 40000]. The performance of these four models is shown in Figures 6 – 9, respectively. It is evident from Figures 6 – 9 that as the size of the training data increases, the performance of the model increases and the difference between the performance on the training and validation data decreases. This difference decreases to the point that the model begins to perform better on the validation data than on the training data. This is evident when the size of the training data is equal to 40000. This is more than likely a result of the addition of dropout because the model is able to generalise better.

Finally, the increase in the performance and the time taken to train the model as the size of the training data increase is shown in Figure 10. It is evident from Figure 10 that both the performance of the model and the time taken to train the model increases linearly as the size of the training data increases. However, it is expected that the performance of the model should plateau as the training data size continues to increase, whereas the time taken to train the model should continue to increase linearly. Consequently, there is a trade-off between improving the performance of the model and the time taken to train that model.

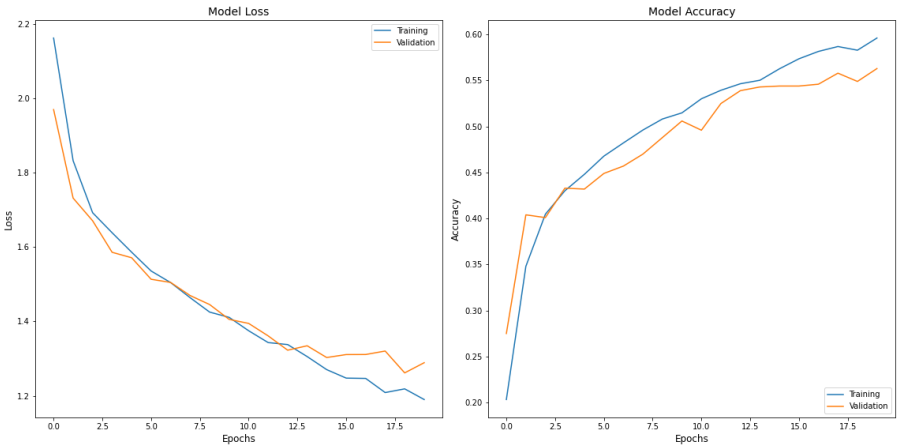

Figure 7. Performance of Model when training data size = 10000

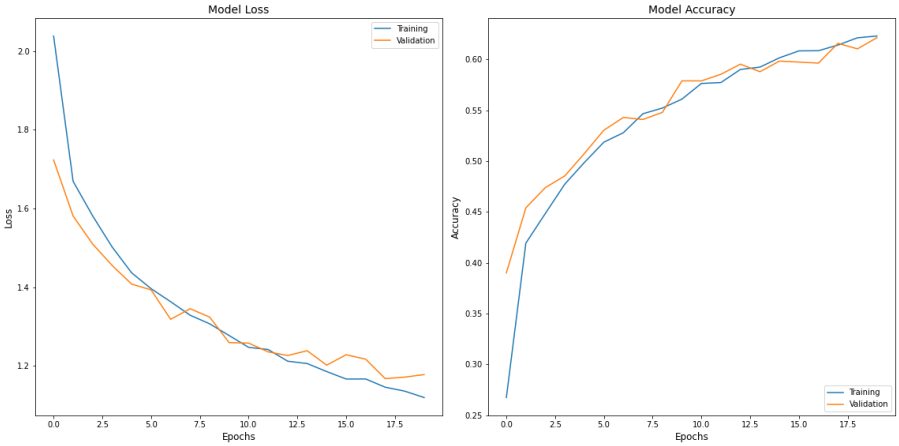

Figure 8. Performance of Model when training data size = 20000

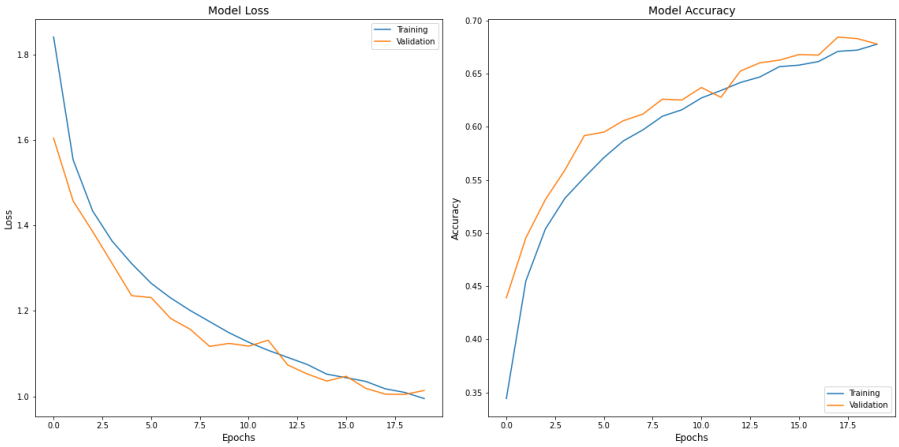

Figure 9. Performance of model when training data size = 40000

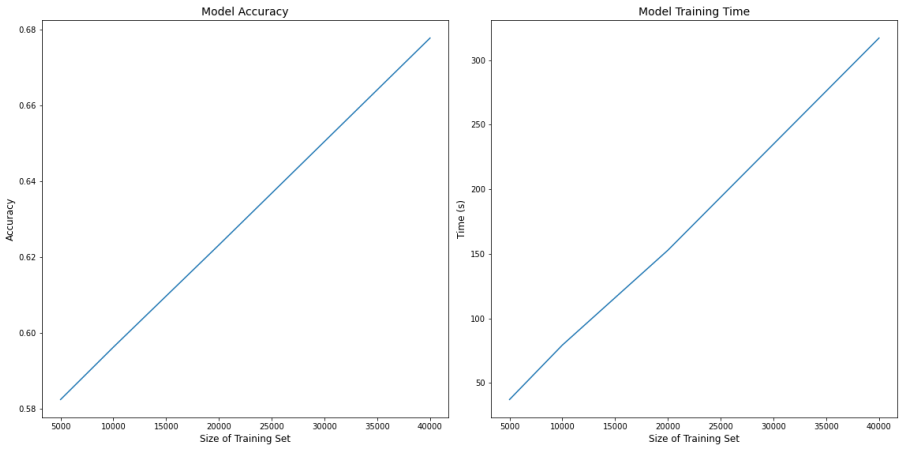

Figure 10. Performance and time taken to train model for a range of training data sizes

(iv) The effect of increasing the L_1 penalty parameter has on the performance of the model was explored. This is shown in Figure 11. It is evident from Figure 11 that as the value of the penalty weight parameter, alpha, increases, the performance of the model deteriorates. This was deduced from the decrease in the prediction accuracy which can be observed in Figure 11. Therefore, the optimal value of the parameter alpha is 0 and other regularisation techniques such as dropout should be explored.

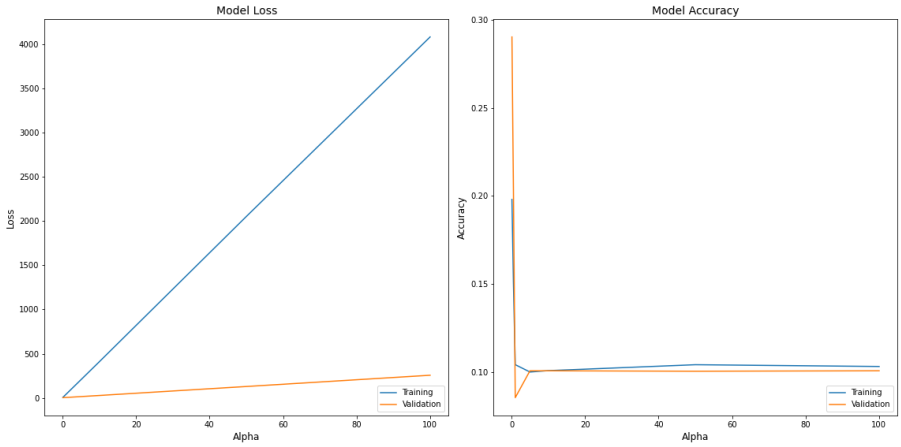

Figure 11. Performance of Model for range of alpha values.

(c) (i) The ConvNet was modified to use max-pooling instead of stride to downsample. This is shown in Figure 12. It is evident from Figure 12 that the stride parameter was removed from the second convolution in line 60, and a max-pooling layer was then added to line 61. Additionally, the stride was removed from the fourth convolutional layer in line 63, and another max-pooling layer was then added to line 64. The two max-pooling layers used a pool size of (2,2) to downsample by (2,2) as well as zero padding.

```
58      model = keras.Sequential()
59      model.add(Conv2D(16, (3,3), padding='same', input_shape=x_train.shape[1:],activation='relu'))
60      model.add(Conv2D(16, (3,3), padding='same', activation='relu'))
61      model.add(MaxPooling2D(pool_size=(2,2), padding='same'))
62      model.add(Conv2D(32, (3,3), padding='same', activation='relu'))
63      model.add(Conv2D(32, (3,3), padding='same', activation='relu'))
64      model.add(MaxPooling2D(pool_size=(2,2), padding='same'))
65      model.add(Dropout(0.5))
66      model.add(Flatten())
67      model.add(Dense(num_classes, activation='softmax',kernel_regularizer=regularizers.l1(0.0001)))
68      model.compile(loss="categorical_crossentropy", optimizer='adam', metrics=["accuracy"])
```

Figure 12. Downsampling using MaxPooling

(ii) The performance of the modified ConvNet was evaluated using 5,000 training data points. It was observed by using the Keras function summary, that the model had 37,146 parameters. This is equal to the number of parameters of the previous model. However, the performance of the model had improved. This is shown in Figure 13. The validation loss and accuracy of both of the models are shown in Figure 13. It is evident from Figure 13 that the accuracy of the model improved, and that the model is less likely to suffer from underfitting. The loss function of the original ConvNet plateaued. However, the loss of the modified ConNet could probably be improved further by increasing the number of epochs.

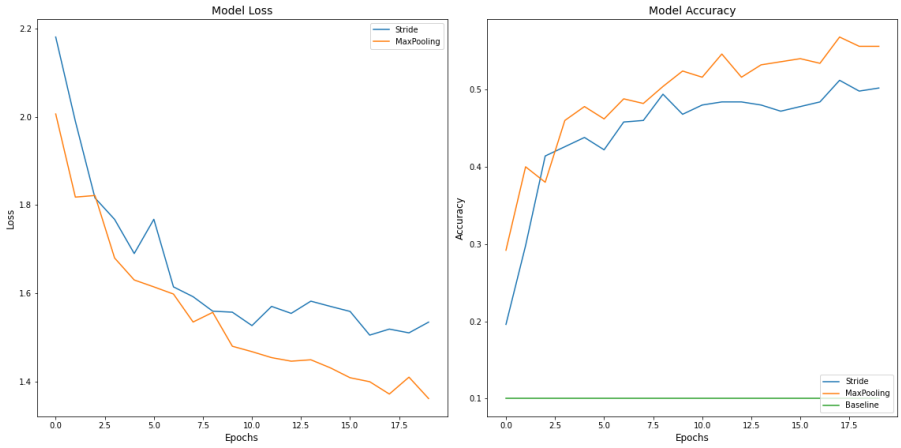

Figure 13. Evaluation of Stride and Max-pooling

Appendix:

Assignment6Part1.py

```
import cv2
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image

def conv2d(array, kernel):
    n, n0 = array.shape
    assert n == n0
    k, k0 = kernel.shape
    assert k == k0

    kernel = np.flipud(np.fliplr(kernel))
    output = np.zeros((n - k + 1, n - k + 1))

    for y in range(output.shape[1]):
        for x in range(output.shape[0]):
            output[y, x] = (kernel * array[y:y+k, x:x+k]).sum()
    return output

img_grey = cv2.imread('tcd.jpg', 0)
fig = plt.figure(figsize=(6, 6))
ax = fig.add_subplot(1, 1, 1)
ax.set_xlabel('Original Image', fontsize=12)
ax.set_xticks([])
ax.set_yticks([])
plt.imshow(img_grey, cmap='gray')
plt.show()

kernel1 = np.array([[-1, -1, -1], [-1, 8, -1], [-1, -1, -1]])
img_conv1 = conv2d(img_grey, kernel=kernel1)
fig = plt.figure(figsize=(6, 6))
ax = fig.add_subplot(1, 1, 1)
ax.set_xlabel('Kernel 1', fontsize=12)
ax.set_xticks([])
ax.set_yticks([])
plt.imshow(img_conv1, cmap='gray')
plt.show()
cv2.imwrite('img_conv1.jpg', img_conv1)

kernel2 = np.array([[0, -1, 0], [-1, 8, -1], [0, -1, 0]])
img_conv2 = conv2d(img_grey, kernel=kernel2)
fig = plt.figure(figsize=(6, 6))
ax = fig.add_subplot(1, 1, 1)
ax.set_xlabel('Kernel 2', fontsize=12)
```

```
ax.set_xticks([])
ax.set_yticks([])
plt.imshow(img_conv2, cmap='gray')
plt.show()
cv2.imwrite('img_conv2.jpg', img_conv2)
```

Assignment6Part1.py

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, regularizers
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten,
BatchNormalization
from tensorflow.keras.layers import Conv2D, MaxPooling2D, LeakyReLU
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
from sklearn.utils import shuffle
from sklearn.dummy import DummyClassifier
import matplotlib.pyplot as plt
import sys
from time import time

num_classes = 10
input_shape = (32, 32, 3)

history_list = []; training_time_list = []
(x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()
n = 5000
x_train = x_train[1:n]; y_train=y_train[1:n]

x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255
print("orig x_train shape:", x_train.shape)

y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

use_saved_model = False
if use_saved_model:
   model = keras.models.load_model("cifar.model")
else:
   model = keras.Sequential()
   model.add(Conv2D(16, (3,3), padding='same',
input_shape=x_train.shape[1:],activation='relu'))
   model.add(Conv2D(16, (3,3), strides=(2,2), padding='same', activation='relu'))
   model.add(Conv2D(32, (3,3), padding='same', activation='relu'))
   model.add(Conv2D(32, (3,3), strides=(2,2), padding='same', activation='relu'))
   model.add(Dropout(0.5))
   model.add(Flatten())
   model.add(Dense(num_classes,
activation='softmax',kernel_regularizer=regularizers.l1(0.0001)))
   model.compile(loss="categorical_crossentropy", optimizer='adam', metrics=["accuracy"])
   model.summary()
   batch_size = 128
```

```
    epochs = 20
    start_time = time()
    history = model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs,
validation_split=0.1)
    training_time = (time() - start_time)
    history_list.append(history)
    training_time_list.append(training_time)
    model.save('cifar.model')

    dummy = DummyClassifier(strategy='most_frequent')
    y_train_flatten = np.argmax(y_train, axis=-1)
    dummy.fit(x_train, y_train_flatten)
    pred = dummy.predict(x_test)
    y_test_flatten = np.argmax(y_test, axis=-1)
    acc_baseline = accuracy_score(pred, y_test_flatten)
    acc_baseline = [acc_baseline] * epochs

    fig = plt.figure(figsize=(16, 8))
    ax = fig.add_subplot(1, 2, 1)
    ax.plot(history.history['loss'])
    ax.plot(history.history['val_loss'])
    ax.set_title('Model Loss', fontsize=14)
    ax.set_ylabel('Loss', fontsize=12)
    ax.set_xlabel('Epochs', fontsize=12)
    ax.legend(['Training', 'Validation', 'Baseline'], loc='upper right')
    ax = fig.add_subplot(1, 2, 2)
    ax.plot(history.history['accuracy'])
    ax.plot(history.history['val_accuracy'])
    ax.plot(acc_baseline)
    ax.set_title('Model Accuracy', fontsize=14)
    ax.set_ylabel('Accuracy', fontsize=12)
    ax.set_xlabel('Epochs', fontsize=12)
    ax.legend(['Training', 'Validation', 'Baseline'], loc='lower right')
    plt.tight_layout()
    plt.savefig('evaluation_against_baseline')
    plt.show()



size_list = [5000, 10000, 20000, 40000]
history_list = []; training_time_list = []
for size in size_list:
    (x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()
    x_train = x_train[1:size]; y_train=y_train[1:size]

    x_train = x_train.astype("float32") / 255
    x_test = x_test.astype("float32") / 255
```

```
    print("orig x_train shape:", x_train.shape)

    y_train = keras.utils.to_categorical(y_train, num_classes)
    y_test = keras.utils.to_categorical(y_test, num_classes)

    use_saved_model = False
    if use_saved_model:
        model = keras.models.load_model("cifar.model")
    else:
        model = keras.Sequential()
        model.add(Conv2D(16, (3,3), padding='same',
input_shape=x_train.shape[1:],activation='relu'))
        model.add(Conv2D(16, (3,3), strides=(2,2), padding='same', activation='relu'))
        model.add(Conv2D(32, (3,3), padding='same', activation='relu'))
        model.add(Conv2D(32, (3,3), strides=(2,2), padding='same', activation='relu'))
        model.add(Dropout(0.5))
        model.add(Flatten())
        model.add(Dense(num_classes,
activation='softmax',kernel_regularizer=regularizers.l1(0.0001)))
        model.compile(loss="categorical_crossentropy", optimizer='adam',
metrics=["accuracy"])
        model.summary()
        batch_size = 128
        epochs = 20
        start_time = time()
        history = model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs,
validation_split=0.1)
        training_time = (time() - start_time)
        history_list.append(history)
        training_time_list.append(training_time)
        model.save('cifar.model')

        fig = plt.figure(figsize=(16, 8))
        ax = fig.add_subplot(1, 2, 1)
        ax.plot(history.history['loss'])
        ax.plot(history.history['val_loss'])
        ax.set_title('Model Loss', fontsize=14)
        ax.set_ylabel('Loss', fontsize=12)
        ax.set_xlabel('Epochs', fontsize=12)
        ax.legend(['Training', 'Validation'], loc='upper right')
        ax = fig.add_subplot(1, 2, 2)
        ax.plot(history.history['accuracy'])
        ax.plot(history.history['val_accuracy'])
        ax.set_title('Model Accuracy', fontsize=14)
        ax.set_ylabel('Accuracy', fontsize=12)
        ax.set_xlabel('Epochs', fontsize=12)
        ax.legend(['Training', 'Validation'], loc='lower right')
```

```
        plt.tight_layout()
        plt.savefig('training_size_' + str(size))
        plt.show()

fig = plt.figure(figsize=(16, 8))
ax = fig.add_subplot(1, 2, 1)
accuracy_list = [];
for history in history_list:
    accuracy_list.append(history.history['accuracy'][-1])
ax.plot(size_list, accuracy_list)
ax.set_title('Model Accuracy', fontsize=14)
ax.set_ylabel('Accuracy', fontsize=12)
ax.set_xlabel('Size of Training Set', fontsize=12)
ax = fig.add_subplot(1, 2, 2)
ax.plot(size_list, training_time_list)
ax.set_title('Model Training Time', fontsize=14)
ax.set_ylabel('Time (s)', fontsize=12)
ax.set_xlabel('Size of Training Set', fontsize=12)
plt.tight_layout()
plt.savefig('evaluation_of_varying_size')
plt.show()



penalty_list = [0, 0.1, 0.5, 1, 5, 10, 50, 100]
history_list = []; training_time_list = []
for penalty in penalty_list:
    (x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()
    x_train = x_train[1:n]; y_train=y_train[1:n]

    x_train = x_train.astype("float32") / 255
    x_test = x_test.astype("float32") / 255
    print("orig x_train shape:", x_train.shape)

    y_train = keras.utils.to_categorical(y_train, num_classes)
    y_test = keras.utils.to_categorical(y_test, num_classes)

    use_saved_model = False
    if use_saved_model:
        model = keras.models.load_model("cifar.model")
    else:
        model = keras.Sequential()
        model.add(Conv2D(16, (3,3), padding='same',
input_shape=x_train.shape[1:],activation='relu'))
        model.add(Conv2D(16, (3,3), strides=(2,2), padding='same', activation='relu'))
        model.add(Conv2D(32, (3,3), padding='same', activation='relu'))
        model.add(Conv2D(32, (3,3), strides=(2,2), padding='same', activation='relu'))
```

```
        model.add(Dropout(0.5))
        model.add(Flatten())
        model.add(Dense(num_classes,
activation='softmax',kernel_regularizer=regularizers.l1(penalty)))
        model.compile(loss="categorical_crossentropy", optimizer='adam',
metrics=["accuracy"])
        model.summary()
        batch_size = 128
        epochs = 20
        start_time = time()
        history = model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs,
validation_split=0.1)
        training_time = (time() - start_time)
        history_list.append(history)
        training_time_list.append(training_time)
        model.save('cifar.model')

loss_list = []; val_loss_list = []
acc_list = []; val_acc_list = []
for history in history_list:
    loss_list.append(history.history['loss'][-1])
    val_loss_list.append(history.history['val_loss'][-1])
    acc_list.append(history.history['accuracy'][-1])
    val_acc_list.append(history.history['val_accuracy'][-1])

fig = plt.figure(figsize=(16, 8))
ax = fig.add_subplot(1, 2, 1)
ax.plot(penalty_list, loss_list)
ax.plot(penalty_list, val_loss_list)
ax.set_title('Model Loss', fontsize=14)
ax.set_ylabel('Loss', fontsize=12)
ax.set_xlabel('Alpha', fontsize=12)
ax.legend(['Training', 'Validation'], loc='lower right')
ax = fig.add_subplot(1, 2, 2)
ax.plot(penalty_list, acc_list)
ax.plot(penalty_list, val_acc_list)
ax.set_title('Model Accuracy', fontsize=14)
ax.set_ylabel('Accuracy', fontsize=12)
ax.set_xlabel('Alpha', fontsize=12)
ax.legend(['Training', 'Validation'], loc='lower right')
plt.tight_layout()
plt.savefig('evaluation_of_varying_penalty')
plt.show()
```

Assignment6Part3.py

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, regularizers
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten,
BatchNormalization
from tensorflow.keras.layers import Conv2D, MaxPooling2D, LeakyReLU
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.utils import shuffle
import matplotlib.pyplot as plt
import sys
from time import time
from sklearn.dummy import DummyClassifier
from sklearn.metrics import accuracy_score

num_classes = 10
input_shape = (32, 32, 3)

history_list = []; training_time_list = []
(x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()
n = 5000
x_train = x_train[1:n]; y_train=y_train[1:n]

x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255
print("orig x_train shape:", x_train.shape)

y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

history_list = []
use_saved_model = False
if use_saved_model:
   model = keras.models.load_model("cifar.model")
else:
   model = keras.Sequential()
   model.add(Conv2D(16, (3,3), padding='same',
input_shape=x_train.shape[1:],activation='relu'))
   model.add(Conv2D(16, (3,3), strides=(2,2), padding='same', activation='relu'))
   model.add(Conv2D(32, (3,3), padding='same', activation='relu'))
   model.add(Conv2D(32, (3,3), strides=(2,2), padding='same', activation='relu'))
   model.add(Dropout(0.5))
   model.add(Flatten())
   model.add(Dense(num_classes,
activation='softmax',kernel_regularizer=regularizers.l1(0.0001)))
   model.compile(loss="categorical_crossentropy", optimizer='adam', metrics=["accuracy"])
```

```
    model.summary()
    batch_size = 128
    epochs = 20
    start_time = time()
    history = model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs,
validation_split=0.1)
    training_time = (time() - start_time)
    history_list.append(history)
    training_time_list.append(training_time)
    model.save('cifar.model')

use_saved_model = False
if use_saved_model:
    model = keras.models.load_model("cifar.model")
else:
    model = keras.Sequential()
    model.add(Conv2D(16, (3,3), padding='same',
input_shape=x_train.shape[1:],activation='relu'))
    model.add(Conv2D(16, (3,3), padding='same', activation='relu'))
    model.add(MaxPooling2D(pool_size=(2,2), padding='same'))
    model.add(Conv2D(32, (3,3), padding='same', activation='relu'))
    model.add(Conv2D(32, (3,3), padding='same', activation='relu'))
    model.add(MaxPooling2D(pool_size=(2,2), padding='same'))
    model.add(Dropout(0.5))
    model.add(Flatten())
    model.add(Dense(num_classes,
activation='softmax',kernel_regularizer=regularizers.l1(0.0001)))
    model.compile(loss="categorical_crossentropy", optimizer='adam', metrics=["accuracy"])
    model.summary()
    batch_size = 128
    epochs = 20
    start_time = time()
    history = model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs,
validation_split=0.1)
    training_time = (time() - start_time)
    history_list.append(history)
    training_time_list.append(training_time)
    model.save('cifar.model')

dummy = DummyClassifier(strategy='most_frequent')
y_train_flatten = np.argmax(y_train, axis=-1)
dummy.fit(x_train, y_train_flatten)
pred = dummy.predict(x_test)
y_test_flatten = np.argmax(y_test, axis=-1)
acc_baseline = accuracy_score(pred, y_test_flatten)
acc_baseline = [acc_baseline] * epochs
```

```
fig = plt.figure(figsize=(16, 8))
ax = fig.add_subplot(1, 2, 1)
alpha_list = [0.25, 1.0]
for history, alpha in zip(history_list, alpha_list):
   ax.plot(history.history['val_loss'])
ax.set_title('Model Loss', fontsize=14)
ax.set_ylabel('Loss', fontsize=12)
ax.set_xlabel('Epochs', fontsize=12)
ax.legend(['Stride', 'MaxPooling', 'Baseline'], loc='upper right')
ax = fig.add_subplot(1, 2, 2)
alpha_list = [0.25, 1.0]
for history, alpha in zip(history_list, alpha_list):
   ax.plot(history.history['val_accuracy'])
ax.plot(acc_baseline)
ax.set_title('Model Accuracy', fontsize=14)
ax.set_ylabel('Accuracy', fontsize=12)
ax.set_xlabel('Epochs', fontsize=12)
ax.legend(['Stride', 'MaxPooling', 'Baseline'], loc='lower right')
plt.tight_layout()
plt.savefig('evaluation_of_maxpooling')
plt.show()
```