Assignment Week 8

Student Name: Patrick Farmer Student Number: 20501828

Date: 23-10-2024

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

I(a)

I began the assignment by implementing a function that convolves a an input array with a kernel and returns the result. The kernel is setup to be a argument to the function. The kernel is setup to iterate through the input array (image) from left to right and top to bottom all the while applying the kernel and saving the output to another array which is eventually returned. The function also does not perform any sort of padding which means that the output array will be smaller than the input array. If the same image is convolved a number of times there may be significant loss of information.

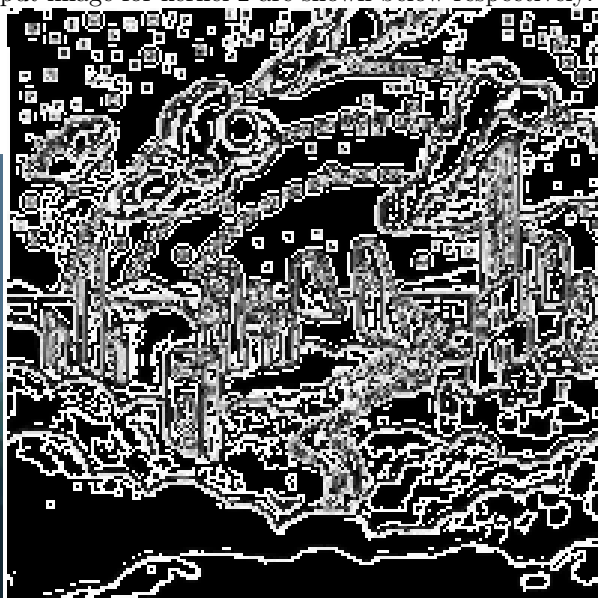
I(b)

This section will implement the above function on an image. The image which I selected was the first I found being my desktop background. The two kernels implemented are shown below:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 8 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

The input image, output image for kernel 1 and output image for kernel 2 are shown below respectively:



II(a)

II(b)(i)

II(b)(ii)

II(b)(iii)

II(b)(iv)

II(c)(i)

II(c)(ii)

II(d)

Appendices

I(a)

```
import numpy as np

def convolve(input_array, kernel):
    n = input_array.shape[0]
    k = kernel.shape[0]
    output_size = n - k + 1
    output_array = np.zeros((output_size, output_size))

    for i in range(output_size):
        for j in range(output_size):
            sub_array = input_array[i:i+k, j:j+k]
            product = sub_array * kernel
            sum_product = np.sum(product)
            output_array[i, j] = sum_product
```

```
    return output_array
```

I(b)

```
import convolution
from PIL import Image
import numpy as np
import os
import week8

downloads_folder = os.path.expanduser('~/Downloads')
image_files = [f for f in os.listdir(downloads_folder) if f.lower().endswith(('.png', '.jpg', '.jpeg'

if not image_files:
    raise FileNotFoundError("No image files found in the downloads folder.")

first_image_path = os.path.join(downloads_folder, image_files[0])
im = Image.open(first_image_path)

width, height = im.size
new_height = 200
new_width = int((new_height / height) * width)
im = im.resize((new_width, new_height), Image.LANCZOS)
rgb = np.array(im.convert('RGB'))

images_folder = os.path.expanduser('Images')
os.makedirs(images_folder, exist_ok=True)
resized_image_path = os.path.join(images_folder, 'original.png')
im.save(resized_image_path)

r = rgb[:, :, 0]

kernel1 = np.array([[-1, -1, -1], [-1, 8, -1], [-1, -1, -1]])
kernel2 = np.array([[0, -1, 0], [-1, 8, -1], [0, -1, 0]])

result1 = convolution.convolve(r, kernel1)
result2 = convolution.convolve(r, kernel2)

result1_image_path = os.path.join(images_folder, 'result1.png')
result2_image_path = os.path.join(images_folder, 'result2.png')

Image.fromarray(np.uint8(result1)).save(result1_image_path)
Image.fromarray(np.uint8(result2)).save(result2_image_path)

week8.run_models()
```

II(a)

```
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
history = model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_split=0.1)
```

II(b)(iii)

```
best_model = None
best_model_description = ""
best_results = model_runner.Results(0, 0, 0, 0)


training_data_sizes = [5000, 10000, 20000, 40000]

for training_data_size in training_data_sizes:
    model_run = model_runner.ModelRunner(x_train_subset, y_train_subset, x_test, y_test, num_classes,
    model_run.train_and_evaluate(training_data_size)
    model_results = model_run.results
    print("\033[93mStrides Model Results:\033[0m", model_results)
    if (model_results.validation_accuracy > best_results.validation_accuracy):
        best_results = model_results
        best_model = model_run.model
        best_model_description = "Strides_" + str(training_data_size) + "_" + str(regularisation_size

class Results(NamedTuple):
    training_time: float
    train_accuracy: float
    test_accuracy: float
    validation_accuracy: float

class ModelRunner:
    def __init__(self, x_train, y_train, x_test, y_test, num_classes, regularisation_size, downsampli
        self.x_train = x_train
        self.y_train = y_train
        self.x_test = x_test
        self.y_test = y_test
        self.num_classes = num_classes
        self.regularisation_size = float(regularisation_size)
        self.downsampling = downsampling
        self.model = self.build_model()
        self.results = Results(training_time=0.0, train_accuracy=0.0, test_accuracy=0.0, validation_a
        self.epochs = epochs

    def build_model(self):
        model = Sequential()
        model.add(Conv2D(16, (3,3), padding='same', input_shape=self.x_train.shape[1:], activation='r
```

```
        if self.downsampling == "strides":
            model.add(Conv2D(16, (3,3), strides=(2,2), padding='same', activation='relu'))
        else:
            model.add(Conv2D(16, (3,3), padding='same', activation='relu'))
            model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Conv2D(32, (3,3), padding='same', activation='relu'))
        if self.downsampling == "strides":
            model.add(Conv2D(32, (3,3), strides=(2,2), padding='same', activation='relu'))
        else:
            model.add(Conv2D(32, (3,3), padding='same', activation='relu'))
            model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Dropout(0.5))
        model.add(Flatten())
        model.add(Dense(self.num_classes, activation='softmax', kernel_regularizer=regularizers.l1(se
        model.compile(loss="categorical_crossentropy", optimizer='adam', metrics=["accuracy"])
        return model

    def train_and_evaluate(self, training_data_size):
        start_time = time.time()
        history = self.model.fit(self.x_train, self.y_train, batch_size=128, epochs=self.epochs, vali
        end_time = time.time()

        training_time = end_time - start_time

        # Evaluate on training data
        train_preds = self.model.predict(self.x_train)
        y_train_pred = np.argmax(train_preds, axis=1)
        y_train_true = np.argmax(self.y_train, axis=1)
        train_accuracy = np.mean(y_train_pred == y_train_true)

        # Evaluate on test data
        test_preds = self.model.predict(self.x_test)
        y_test_pred = np.argmax(test_preds, axis=1)
        y_test_true = np.argmax(self.y_test, axis=1)
        test_accuracy = np.mean(y_test_pred == y_test_true)

        self.results = Results(training_time=training_time, train_accuracy=train_accuracy, test_accur

        # Plotting accuracy and loss
        plt.figure(figsize=(12, 6))
        plt.subplot(211)
        plt.plot(history.history['accuracy'])
        plt.plot(history.history['val_accuracy'])
        plt.title(f'Model accuracy for {training_data_size} samples with regularisation {self.regular
        plt.ylabel('accuracy')
        plt.xlabel('epoch')
        plt.legend(['train', 'val'], loc='upper left')
        plt.subplot(212)
        plt.plot(history.history['loss'])
        plt.plot(history.history['val_loss'])
        plt.title(f'Model loss for {training_data_size} samples with regularisation {self.regularisat
        plt.ylabel('loss')
```

```
        plt.xlabel('epoch')
        plt.legend(['train', 'val'], loc='upper left')
        plt.savefig(f"Images/cifar_{training_data_size}_{self.regularisation_size}_{self.downsampling
        plt.close()

    def get_results(self):
        return self.results
```

II(b)(iv)

```
for training_data_size in training_data_sizes:
    if training_data_size == 5000:
        regularisation_sizes = [0.0001, 0.001, 0.01, 0.1, 0, 1]
    else:
        regularisation_sizes = [0.001]
    for regularisation_size in regularisation_sizes:
        # Same code
```

II(c)(i)

```
model_run = model_runner.ModelRunner(x_train_subset, y_train_subset, x_test, y_test, num_classes, reg
model_run.train_and_evaluate(training_data_size)
model_results = model_run.results
print("\033[93mPooling Model Results:\033[0m", model_results)
if (model_results.validation_accuracy > best_results.validation_accuracy):
    best_results = model_results
    best_model = model_run.model
    best_model_description = "Pooling_" + str(training_data_size) + "_" + str(regularisation_size)

class ModelRunner:
    def build_model(self):
        model = Sequential()
        model.add(Conv2D(16, (3,3), padding='same', input_shape=self.x_train.shape[1:], activation='r
        if self.downsampling == "strides":
            model.add(Conv2D(16, (3,3), strides=(2,2), padding='same', activation='relu'))
        else:
            model.add(Conv2D(16, (3,3), padding='same', activation='relu'))
            model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Conv2D(32, (3,3), padding='same', activation='relu'))
        if self.downsampling == "strides":
            model.add(Conv2D(32, (3,3), strides=(2,2), padding='same', activation='relu'))
        else:
            model.add(Conv2D(32, (3,3), padding='same', activation='relu'))
            model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Dropout(0.5))
        model.add(Flatten())
        model.add(Dense(self.num_classes, activation='softmax', kernel_regularizer=regularizers.l1(se
        model.compile(loss="categorical_crossentropy", optimizer='adam', metrics=["accuracy"])
        return model
```

II(d)

```
model_run = model_runner.ModelRunner(x_train_subset, y_train_subset, x_test, y_test, num_classes, reg
model_run.build_advanced_model()
model_run.train_and_evaluate(training_data_size)
model_results = model_run.results
print("\033[93mAdvanced Model Results:\033[0m", model_results)
if (model_results.validation_accuracy > best_results.validation_accuracy):
    best_results = model_results
    best_model = model_run.model
    best_model_description = "Advanced_" + str(training_data_size) + "_" + str(regularisation_size)

def build_advanced_model(self):
    model = Sequential()
    model.add(Conv2D(8, (3, 3), padding='same', input_shape=self.x_train.shape[1:], activation='relu'
    model.add(Conv2D(8, (3, 3), strides=(2, 2), padding='same', activation='relu'))
    model.add(Conv2D(16, (3, 3), padding='same', 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(self.num_classes, activation='softmax', kernel_regularizer=regularizers.l1(self.r
    model.compile(loss="categorical_crossentropy", optimizer='adam', metrics=["accuracy"])
    self.model = model
```