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.

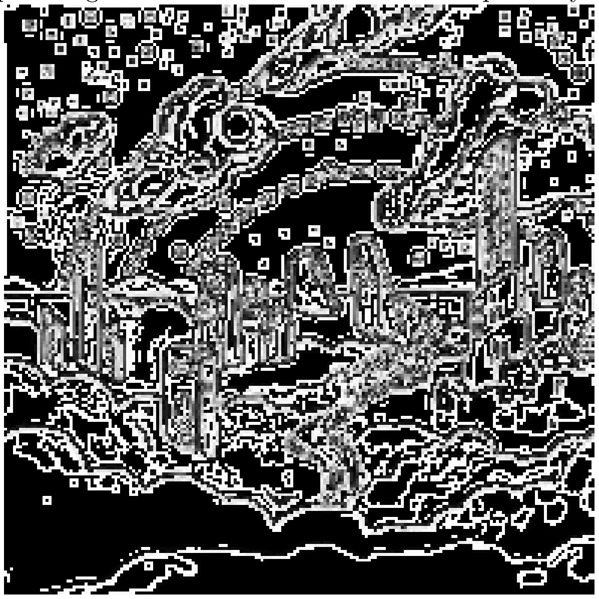
I(b)

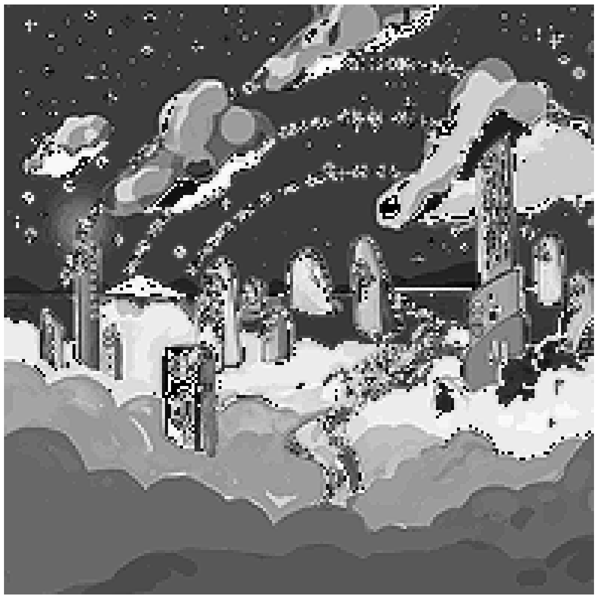
This section will implement the above function on an image. The image which I selected was the first I found being my desktop background. The Image is first resized to 200x200 and then loaded as three RGB arrays then only the red channel taken. The two kernels implemented are shown below:

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

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

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





The first image appears to show an edge image meaning that the kernel is an edge filter.
The second image appears to show a sharpened image as can be seen by the increase in contrast between edges but also the overemphasis of noise. This means that the kernel is a sharpening filter.

II(a)

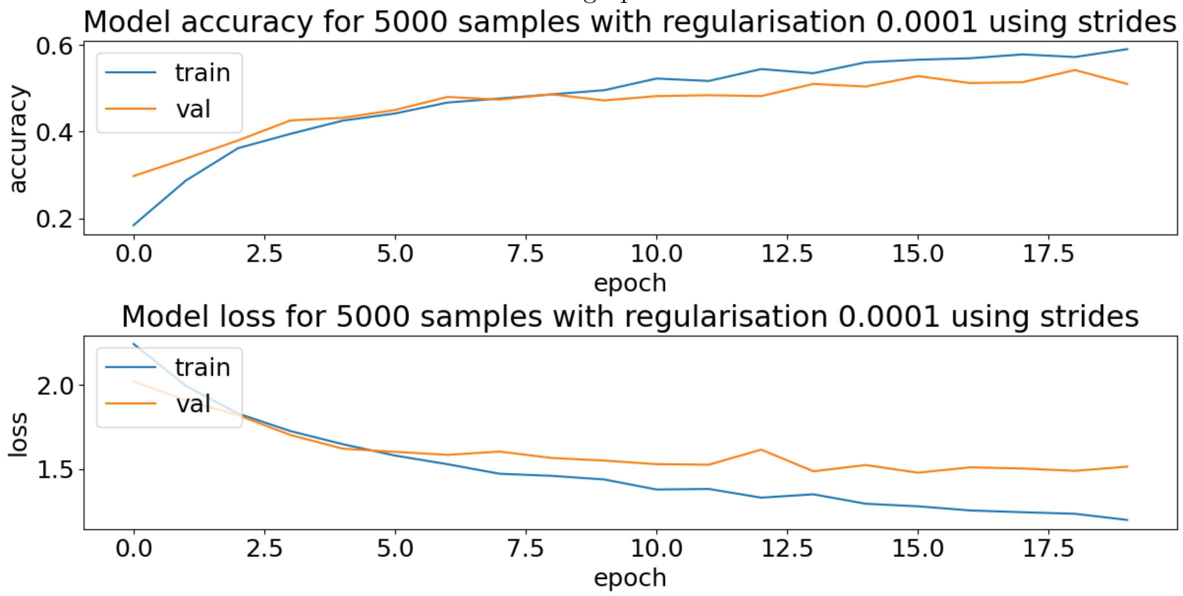
The convolutional network architecture consists of four convolution layers with ReLU activation. Where the first two layers have 16 filters each while the other two have 32 filters each. The kernel size does not change between layers all of them have a 3x3 kernel size, with padding being used which means that after convolutional the original image size is recovered by mirroring the outermost neighbour pixel. The second and fourth layers implement a stride of 2x2 which represents how far across the image the kernel is moved before it is applied. The regularisation that is used by this model is first a dropout layer of 0.5, this means that for each epoch half of the neurons will be shut off so that the model does not just learn a couple ‘paths’ to make it’s decision. The second form of regularisation used is L1 regularisation with a value of 0.0001 which adds a penalty to the loss function for large weights. This is used to prevent overfitting.
The model is then flattened into a 2d array. Finally a dense layer is applied with a softmax activation function. Each neuron is connected to every neuron in the previous layer. The amount of neurons matches the amount of output classes which in this case is 10. The softmax function is used to convert the output of the model into a probability distribution.
The batch size is 128 and the model is trained for 20 epochs.

II(b)(i)

The model when run with 5000 data points does get an accuracy close to 48% on the test data. In this can 51%. The training data performs much better with 63%. The model is overfitting but there is also few improvements to be made by increasing regularisation, the data set would need to be larger for more substantial improvements.
The amount of parameters in this mode is 37,146 parameters and then 74,294 optimiser parameters. The layer with the largest parameter count is the dense layer with 20,490 parameters.
When comparing against a baseline classifier that simply predicts the most common class the model is performing much better. The baseline classifier has an accuracy of 10% which means that the dataset is balanced amongst the classes otherwise the accuracy of baseline would be higher than 100%/10.

II(b)(ii)

The validation accuracy and training accuracy of the model can be plotted over its epochs along with the loss of the model. This is shown in the below graph:

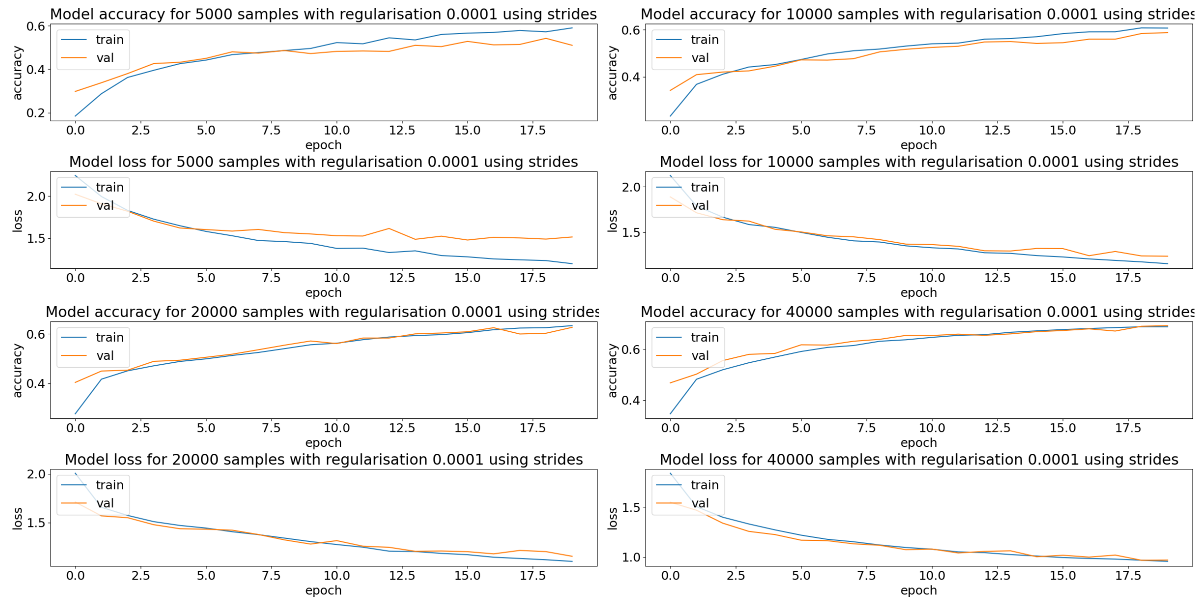


As already stated by the difference in the training and validation accuracy we can see that this model is overfitting. From the graph we can tell this because the line for the training accuracy and validation accuracy begin to diverge from one another before the end of the training. We can also see it from the loss graph because the training loss continues to decrease while the validation loss becomes stagnant.

II(b)(iii)

Training Data Size	Training Time (s)	Train Accuracy	Test Accuracy	Validation Accuracy
5000	4.06	0.60	0.49	0.48
10000	6.85	0.67	0.57	0.58
20000	13.47	0.69	0.62	0.63
40000	24.82	0.74	0.69	0.69

The data shown in the table above shows that as the training data size increases the test accuracy increase. Noteably as previously mentioned also the model is overfitting less as the training data size increases. This is due to the increase in diversity of the training data and improves the models ability to generalise.
As the training data size increases the training time also predictably increases. In this case the training was done on a CPU so all of the training times are long. We can see that the training time for each doubling of the training data size is between 75% and 100% longer.
The graphs below show the same data as the table above, we can see that the accuracy increases with the training data size as does the effects of overfitting lessen.



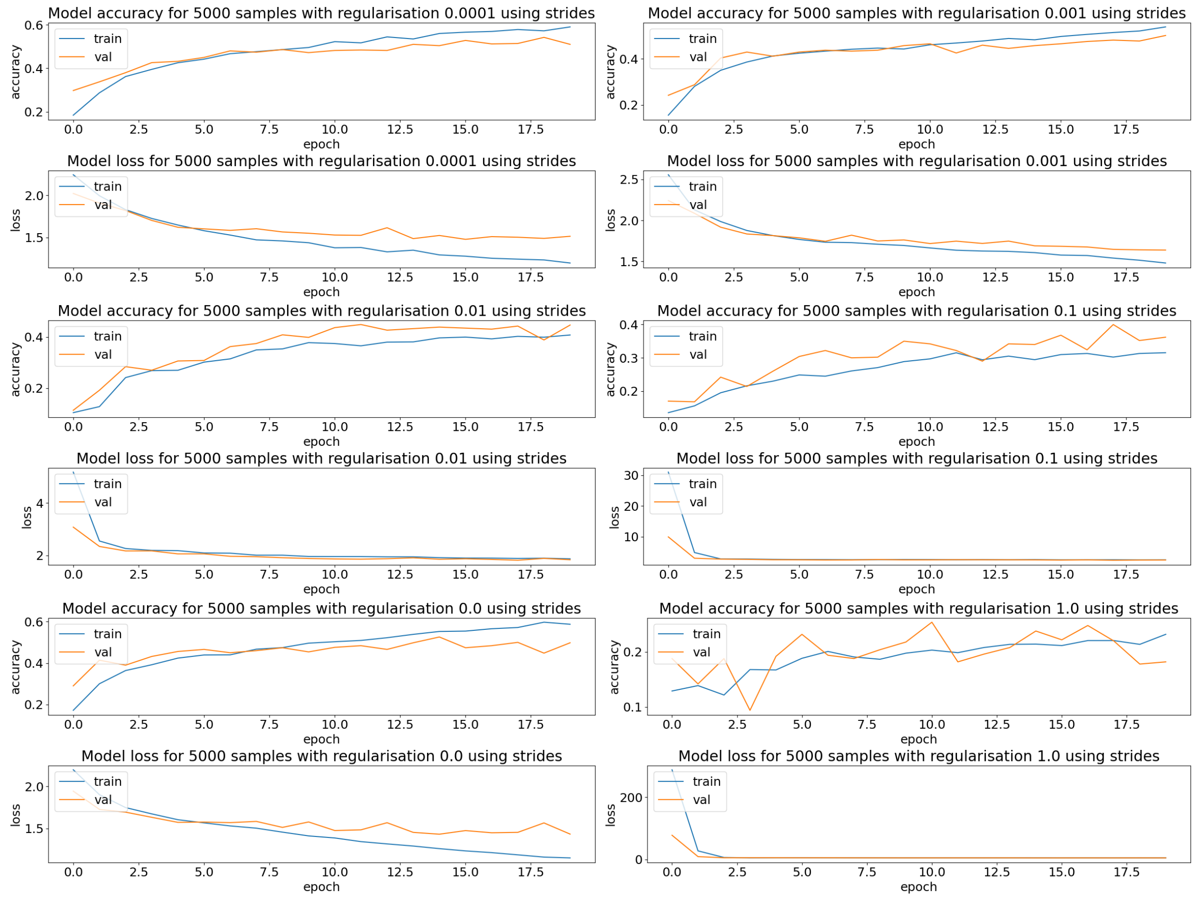
II(b)(iv)

Regularisation Size	Training Time (s)	Train Accuracy	Test Accuracy	Validation Accuracy
0.0001	4.06	0.60	0.49	0.48
0.001	3.92	0.57	0.48	0.48
0.01	3.90	0.46	0.43	0.41
0.1	4.07	0.37	0.35	0.39
0	3.89	0.61	0.49	0.49
1	3.88	0.22	0.21	0.21

The L1 regularisation limits the change in the weights of the model from epoch to epoch. This is done by adding a penalty to the loss function for large weights. The regularisation weight is the size of the penalty applied. This prevents one feature from dominating the model and overfitting which makes the model better at generalising. It will however also slow down the training of the model as the model is not able to learn as quickly.

Up until now the weight of the regularisation was kept constant at 0.0001. In this section the regularisation weight is varied with a set 5000 training data size. We can see that with the increase in regularisation the training accuracy decreases substantially. The validation accuracy does also decrease but not at the same rate. This is because the model is overfitting less and as can be seen from the graph the validation and training accuracy follow the same trend for the higher values of regularisation. 0 is an exception to this as it means that no regularisation is applied, that it why its results are so similar to 0.0001 as this is where the regularisation is the most modest and closest to not being applied. We can see this in the below plots as the two trends of validation and training accuracy and loss are very close for the higher values of regularisation but begin to diverge quickly for the lower values of regularisation.

We can see in this case there is little benefit to increasing the regularisation weight as there is just simply not enough data to achieve a much more accurate model as the models seen in the previous section. The ideal regularisation weight though likely lies between 0.001 and 0.01 as from the graph we can see from the excessive fluctuating of the validation accuracy for 0.01 that the model is underfitting but for the 0.001 regularisation model we can see the training and validation accuracy diverging towards the end meaning that this model is overfitting.



II(c)(i)

Pooling and strides are both methods of downsampling the image. Pooling will take the maximum value of a kernel and apply it to the output while strides will move the kernel across the image by a set amount. The pooling method is likely to achieve better accuracy but is also likely to take longer being more expensive.

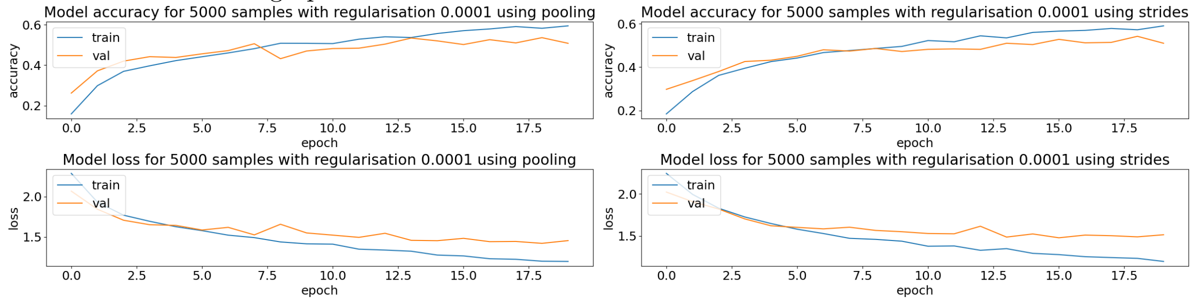
II(c)(ii)

Model Type	Training Data Size	Training Time (s)	Train Accuracy	Test Accuracy	Validation Accuracy
Pooling	40000	40.36	0.76285	0.7164	0.7105
Strides	40000	24.82	0.738725	0.6853	0.6875
Pooling	20000	20.82	0.75205	0.6873	0.6830
Strides	20000	13.47	0.6866	0.6226	0.6280
Pooling	10000	10.82	0.7024	0.6032	0.5800
Strides	10000	6.85	0.6665	0.5717	0.5830

Model Type	Training Data Size	Training Time (s)	Train Accuracy	Test Accuracy	Validation Accuracy
Pooling	5000	5.77	0.6802	0.5589	0.5760
Strides	5000	4.06	0.5976	0.4891	0.4800

Across the board we can see an improvement in the test and validation accuracy in pooling against strides with a new highest value for accuracy with an accuracy of 71.64% for the 40000 training data size model. The most notable improvement is in the 5000 training data size model where the test accuracy goes from 49% to 56%. The training time for the pooling model is higher than the strides model. The difference in training time varies greatly with the data set size but can be as little as ~40% and up to ~80%. The training time is longer for the pooling model as it has to perform its computation more times than the strides downsampling as the strides will simply 'skip' some of the pixels as an origin.
This model has the same amount of parameters as the strides model for both the model parameters of 37,146 and the optimiser parameters of 74,294. The performance difference is not due to extra complexity in the model but a better setting of the hyperparameters.

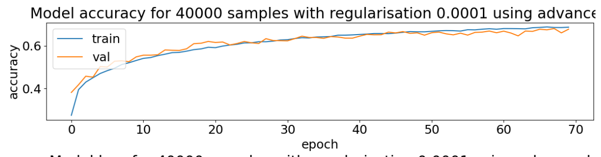
The difference between the two methods are most pronounced in the 5000 training data size model so we will take these two graphs below:

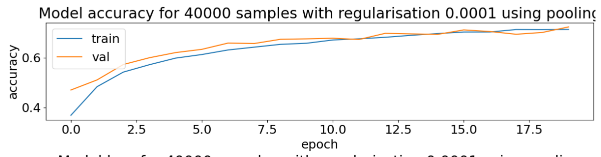


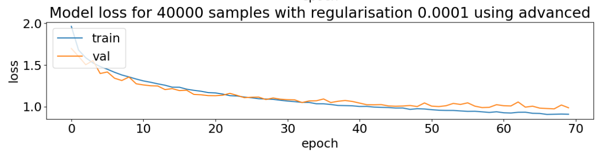
II(d)

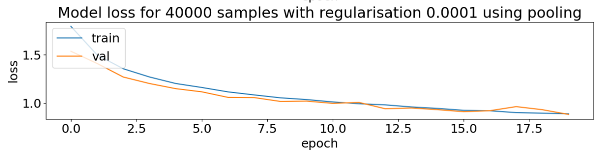
Model Type	Training Data Size	Regularisation Size	Training Time (s)	Train Accuracy	Test Accuracy	Validation Accuracy
Advanced	40000	0.0001	60.87	0.7525	0.677	0.6777
Pooling	40000	0.0001	40.52	0.7741	0.726	0.7243
Strides	40000	0.0001	24.87	0.7444	0.6925	0.6927

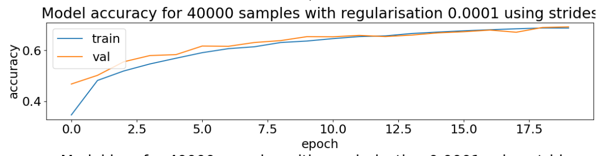
This section involves implementing a more advanced model to try an achieve better results. The model given in the labsheet was taken however, suprisingly this model did not manage to outperform the current most accurate model of pooling at 40000 data points and regularisation of 0.0001 even with substantially more epochs added (70), it also performed worse than the strides model. The accuracy of the model can be seen in the table above and the graph below under the 'Advanced' model type.
The effects of overfitting although not shown in the above table are reduced in the advanced model, the training accuracy being so far above is due to the amount of epochs that the model was trained for a more telling story is the graphs which can be seen below.

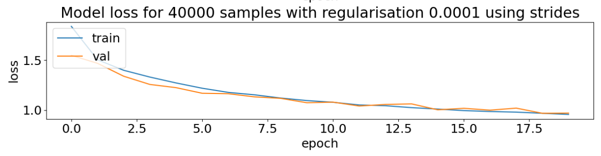












Appendices

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)(i)

```
class_counts = np.bincount(np.argmax(y_train, axis=1))
most_common_class = np.argmax(class_counts)
y_pred_baseline = np.full(y_test.shape[0], most_common_class)
y_pred_baseline = keras.utils.to_categorical(y_pred_baseline, num_classes)

baseline_accuracy = np.mean(np.argmax(y_test, axis=1) == np.argmax(y_pred_baseline, axis=1)) * 100
print("\033[93mBaseline Classifier Accuracy:\033[0m {:.2f}%".format(baseline_accuracy))
```

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
```