Machine Learning Week 8 Assignment

CS7CS4/CSU44061 - Machine Learning - Cormac Madden

i)

a) My convolution function takes an image and kernel as parameters and returns the image with the kernel applied to it. The function creates an output array of zero values, the same size as the input image.
It then calls a separate function I created to pad the input image.

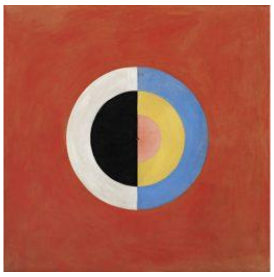
The padding function takes the image and a kernel as input parameters and returns the image with padding half the width of the kernel on each side. The padding is made up of the edge pixels duplicated outwards.

The next step in the convolution function is to loop through the input image and for each pixel:
Create regions of interest (ROI) the size of the kernel
Multiply the kernel by the ROI
Sum the resultant matrix
Set the given pixel equal to the sum of the matrix

I then clip the output between 0 and 255 and then return the image as a unit8.

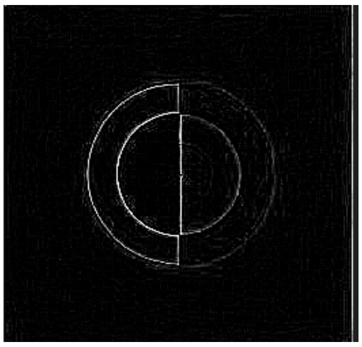
b) After passing the 256x256 abstract art image and shown kernels into the convolution function, the following images are returned.

Input Image

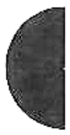


$$kernel1 = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \quad kernel2 = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 8 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Output from kernel1



Output from kernel2



The first kernel is often used for edge detection and the second kernel is used for sharpening.

ii)

a)

Layer	Feature Map	Size	Kernel Size	Stride	Activation
Image	1	32 x 32	-	-	-
Convolution	16	32 x 32 x 16	3x3	1	relu
Convolution	16	16 x 16 x 16	3x3	2	relu
Convolution	32	16 x 16 x 32	3x3	1	relu
Convolution	32	8 x 8 x 32	3x3	2	relu
Dropout	-	8 x 8 x 32	-	-	-
Flatten (FC)	-	-	-	-	-
Dense	-	-	-	-	Softmax

This convolutional network has 7 layers. The first 4 layers all use 3x3 sized kernels. They alternate between convolution layers with one or two strides. The convolutional layers with a stride of 1 increase the dimensionality of the output. The layers with a stride of 2 reduce the dimensionality of the output space, halving the width and height of each output. The Dropout layer is the fifth layer and it is a mask that removes the contribution of, in this case half, the nodes towards the next layer and leaves unmodified all others. It's used to prevent overfitting.

The flatten layer creates a Fully Connected Layer (FCL), by reducing all the features to one dimension.

b)

i)

This Keras model has 37,146 total parameters.

The dense layer has the most parameters, this is because it receives input from all the nodes of the previous layer, which has just been flattened, and multiplies it by the number or classes. It also adds the number of input classes to return 20,490 parameters in this case.

(2048 * 10) + 10 = 20,490

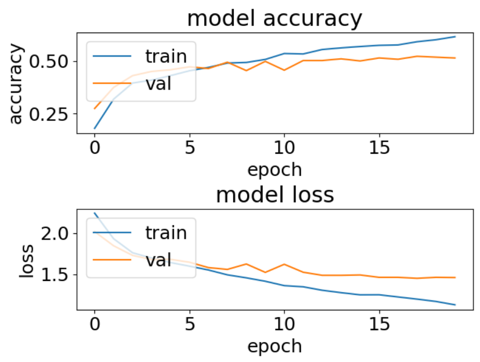
The performance on the test data compares with the training data
The training data returns an accuracy of 0.62, and the accuracy of the test data returns an accuracy of 0.51.

This discrepancy is as expected as the model is based on the training data and it has never seen the testing data before.

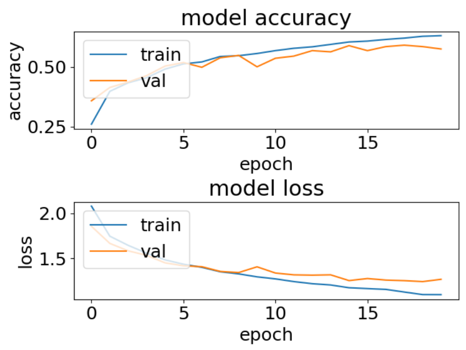
ii)
The plot of the history data shows how the variables such as the accuracy and loss amount, for the training and validation data varies over time. The validation data is separate from the training set and is used to validate our model performance during training.
The purpose of splitting the dataset into a validation set is to prevent the model from overfitting. The model may perform well classifying the samples in the training set but may not be able to generalise and make accurate classifications on unseen data.

iii)

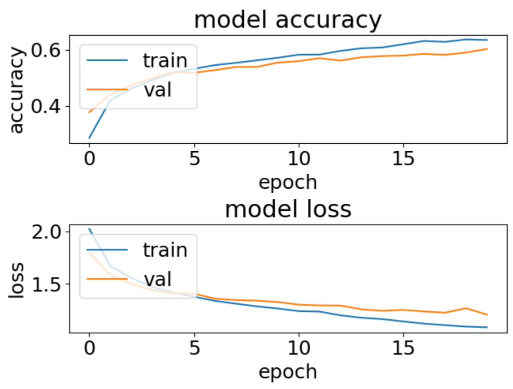
5K, time = 29.84 seconds



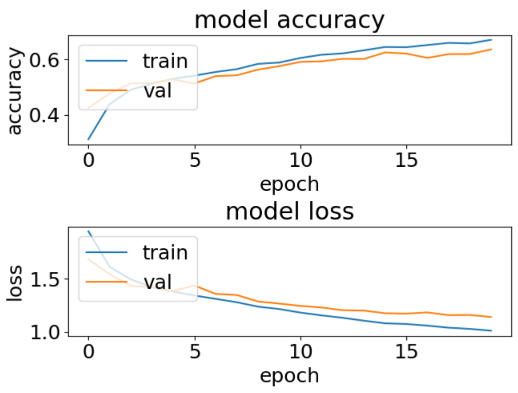
10K, time = 58.82 seconds



15K, time = 87.41 seconds



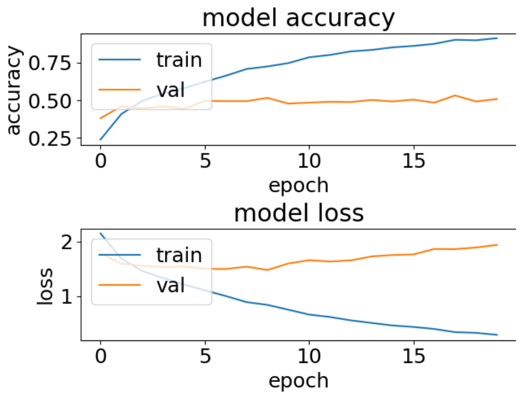
20K, time = 115.83 seconds



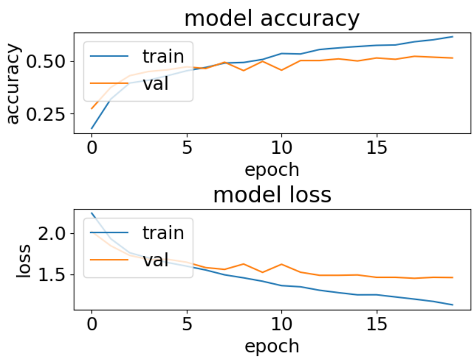
As the training data is increased the noise in the validation curve is reduced.
The gap between the validation data and the training data decreases for both the loss amount and the accuracy. The overall accuracy is increased with more training data and the loss amount is reduced. The increase in time also seems to be linear, with about 30 seconds added for every 5K data points added.

iv)

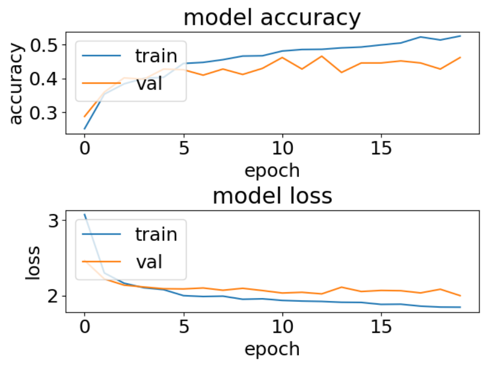
$L_1 = 0.0$



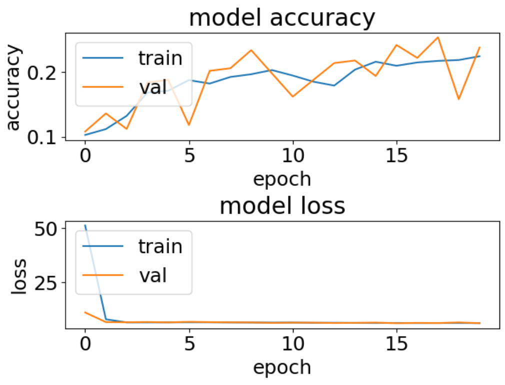
$L_1 = 0.0001$



$L_1 = 0.001$



$L_1 = 0.1$

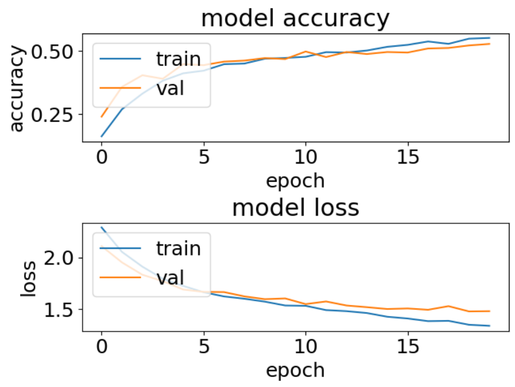


As the weight parameter increases, the accuracy decreases and the model becomes more noisy. When the weight parameter is too low the model overfits the data and results in a high discrepancy between the accuracy of the model on the training data versus the validation data.

Varying the weight is more effective at managing overfitting. Increasing the training data enables the ability to have a more accurate generalised model, but it only indirectly impacts how tightly fitted the model is to the training data.
The weight parameter gives direct control as to how closely fit the model is to the data.

c) i)

The results from MaxPooling:



ii)

After adding two max pooling and replacing the stride layers with another non-stride convolutional layers, the total parameters stayed at 37,146. The two operations cancel each other out resulting in the same number of parameters.

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
#from tensorflow.keras import layers, regularizers
from keras.layers import Dense, Dropout, Activation, Flatten,
BatchNormalization
from keras.layers import Conv2D, MaxPooling2D, LeakyReLU
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.utils import shuffle
import matplotlib.pyplot as plt
plt.rc('font', size=18)
plt.rcParams['figure.constrained_layout.use'] = True
import sys
import time


# Model / data parameters
num_classes = 10
input_shape = (32, 32, 3)

# the data, split between train and test sets
(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]

# Scale images to the [0, 1] range
x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255
print("orig x_train shape:", x_train.shape)

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

start_time = time.time()

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((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((2, 2), padding='same'))
    model.add(Dropout(0.5))
    model.add(Flatten())
    model.add(Dense(num_classes,
activation='softmax',kernel_regularizer=tf.keras.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)
    print("--- %s seconds ---" % (time.time() - start_time))
    model.save("cifar.model")
    plt.subplot(211)
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('model accuracy')
    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('model loss')
    plt.ylabel('loss'); plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')
    plt.show()

preds = model.predict(x_train)
y_pred = np.argmax(preds, axis=1)
y_train1 = np.argmax(y_train, axis=1)
print(classification_report(y_train1, y_pred))
print(confusion_matrix(y_train1,y_pred))

preds = model.predict(x_test)
y_pred = np.argmax(preds, axis=1)
y_test1 = np.argmax(y_test, axis=1)
print(classification_report(y_test1, y_pred))
print(confusion_matrix(y_test1,y_pred))
```

CONVOLUTION CODE

```
import numpy as np
from PIL import Image
im = Image.open('abstract art.jpeg')
rgb = np.array(im.convert('RGB'))
r=rgb[:,:,0] #array of R pixels

def convolution2D(image, kernel):
    output = np.zeros(image.shape)
    padded_image = pad_image(image, kernel)
    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            roi = padded_image[i:i + kernel.shape[0], j:j +
kernel.shape[1]]
            k1 = (kernel * roi)
            k2 = np.sum(k1, axis = 0)
            k3 = np.sum(k2, axis = 0)
            k3 = np.clip(k3,0,255)
            output[i][j]=k3
    output = np.clip(output,0,255)
    output = output.astype('uint8')
    return output

def pad_image(image,kernel):

    image_width = image.shape[0]
    image_height = image.shape[1]
    padx = (kernel.shape[0] - 1) // 2
    pady = (kernel.shape[1] - 1) // 2
    #blank image that's padded on x and y
    if(len(image.shape) > 2):
        padded_image = np.zeros((image_width + (2 * pady), image_height +
(2 * padx),image.shape[2]))
    else:
        padded_image = np.zeros((image_width + (2 * pady), image_height +
(2 * padx)))
    #filled image that's padded on x and y
    padded_image[padx:padded_image.shape[0] - padx,
pady:padded_image.shape[1] - pady] = image
    for i in range(padx):
        #left padding
        padded_image[i,pady:padded_image.shape[1]-pady] = image[0,:]
        #right padding
        padded_image[(i+image_width+padx),pady:padded_image.shape[1]-pady]
= image[image_width-4,:]
    for j in range(pady):
        #top padding
        padded_image[padx:padded_image.shape[0]-padx,j] = image[:,0]
        #bottom padding
        padded_image[padx:padded_image.shape[0]-padx,j+image_height+pady] =
image[:,image_height-4]
    padded_image = padded_image.astype('uint8')
    return padded_image

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

r1 = convolution2D(r,kernel1)
Image.fromarray(np.uint8(r1)).show()

kernel2 = np.array([[0,-1,0],[-1,8,-1],[0,-1,0]])
r2 = convolution2D(r,kernel2)
Image.fromarray(np.uint8(r2)).show()
```