Homework 1

CNN and CIFAR-10

Juan Tarrat

1 Introduction

In this assignment, we were asked to build and test a DL learning algorithm and test it in one of the two datasets MNIST or CIFAR-10. My choice has been a CNN (Convolutional Neural Network) on CIFAR-10 dataset. I implemented mine in Python using TensorFlow's *Keras* package.

In order to get as much accuracy as possible, I did some research on successful architectures to base my design off of them. *AlexNet* [1] is the one I saw that has been very relevant in AI literature. However, the CIFAR-10 is made up of 32 x 32 x 3 images and *AlexNet* uses a 224 x 224 x 3 image as input. Furthermore, CIFAR-10 is used for a 10-class classification problem, whereas AlexNet is used for 1000-class classification problem.

1.1 CNN Algorithm

CNNs are used for classification problems. By using mathematical tools like convolution and pooling, the network is able to reduce the size of the input before feeding it into a fully connected network. In addition to this, with convolution what the network does is feature extraction, which is very helpful for it to identify key details of each of the classes and thus learn to classify them.

1.2 CNN Architecture

The architeture of the network consists of 5 convolutional layers, all of which use $reLu$ as their activation function, a kernel size of 3x3, and 64 and 128 filters. There are also three max pooling layers. After the convolutional layer, a flattening layer is added that reduces the dimensionality of the current input so that we can feed it forward in the fully connected layer. The fully connected section consists of 2 hidden layers of 512 neurons each, which ultimately leads to the ouput layer with 10 neurons. Each dense layer uses $reLu$ except for the output layer, which uses $softmax$.

The dropout layers serve as a way of preventing over fitting. Over fitting occurs when the network learns the training set very well but doe not perform nearly as good with the test set. By adding dropout, we are deactivating a portion of the nodes and thus preventing the problem. Another solution to over fitting not explored in this assignment is data augmentation.

To choose the values for each of the layers, I based myself on the design of [1] and [2]. For the learning rate I used the Adam optimizer provided by Keras. On default mode, the learning rate is 0.001. However, I made a second run of the model with the learning rate being 0.005. The reason for this is that it seemed to me like the progress on each epoch was a little slow, so by incrementing the learning rate what happens is that the updated weights and biases change on a greater amount.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 30, 64)	1792
conv2d (Conv2D)	(None, 28, 28, 64)	36928
max_pooling2 (MaxPooling2D)	(None, 14, 14, 64)	0
dropout (Dropout)	(None, 14, 14, 64)	0
conv2d (Conv2D)	(None, 12, 12, 128)	73856
max_pooling2d (MaxPooling2D)	(None, 6, 6, 128)	0
dropout (Dropout)	(None, 6, 6, 128)	0
conv2d (Conv2D)	(None, 4, 4, 128)	147584
conv2d (Conv2D)	(None, 2, 2, 128)	147584
max_pooling2d (MaxPooling2d)	(None, 1, 1, 128)	0
dropout (Dropout)	(None, 1, 1, 128)	0
flatten (Flatten)	(None, 128)	0
dense (Dense)	(None, 512)	66048
dropout (Dropout)	(None, 512)	0
dense (Dense)	(None, 512)	262656
dropout (Dropout)	(None, 512)	0
dense (Dense)	(None, 10)	5130

2 The Dataset

I selected CIFAR-10 dataset. This dataset is used for image classification problems and consists of $60,000$ 32x32 coloured images. There are $50,000$ training images and $10,000$ test images. In total

there are 10 different classes:

- Airplane
- Automobile
- Bird
- Cat
- Deer
- Dog
- Frog
- Horse
- Ship
- Truck

Keras was used to get the dataset with the following code:

```
from keras.datasets import cifar10
(xTrain, yTrain), (xTest, yTest) = cifar10.load_data()


#get labels for each class
cifar10Labels = {
 0: "Plane",
 1: "Car",
 2: "Bird",
 3: "Cat",
 4: "Deer",
 5: "Dog",
 6: "Frog",
 7: "Horse",
 8: "Boat",
 9: "Truck"
}
```

The dictionary is used to easily access the string for each of the classes when plotting. The next figure shows 9 randomly selected images with their correspondent label, see Figure 1 for sample images.

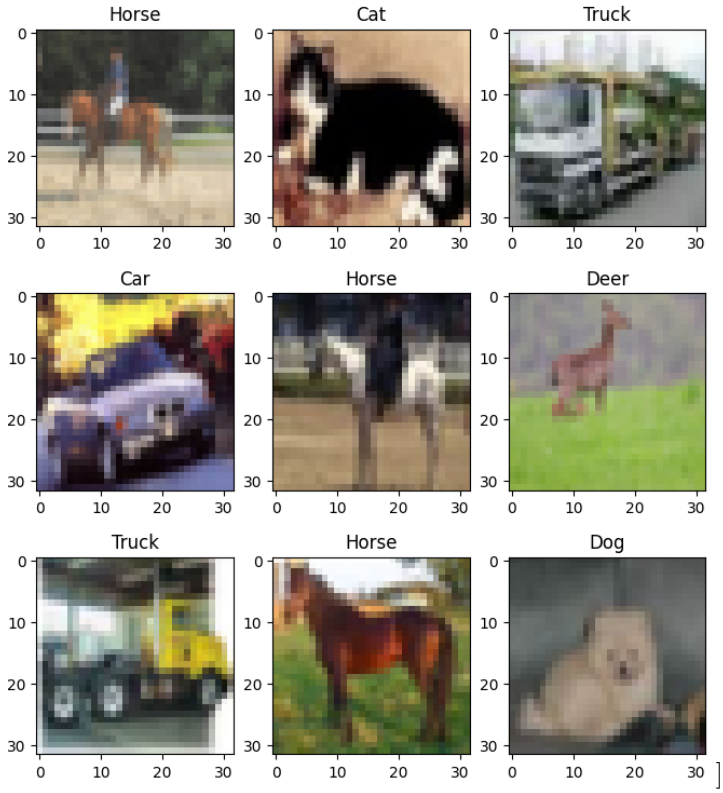

Figure 1: Sample of 9 randomly selected images from the dataset

3 The code

Once the each of the images and labels are stored in their correspondent tuples, some processing needs to be done.

```
#normalize data
xTrain = xTrain.astype('float32')
xTest = xTest.astype('float32')
xTrain /= 255
xTest /= 255


import keras.utils
#create one-hot encoding for labels
yTrainOneHot = keras.utils.to_categorical(yTrain,10)
yTestOneHot = keras.utils.to_categorical(yTest, 10)
```

What the previous snippet does is first transforms the data to floating point type to then normalize it. After that, a Keras utility is done to transform the labels into one-hot encoded vectors. These are 10-item vectors that will be all 0s except for the index that corresponds to the class. This is useful

when outputting the prediction in the output layer in order to classify the image. Next, the model to be used is define using the following code:

```
#create neural network
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D
from keras.constraints import MaxNorm

model1 = Sequential()
model1.add(Conv2D(64,(3,3), input_shape=(32,32,3), activation='relu', kernel_constraint=MaxNorm(3
model1.add(Conv2D(64,(3,3), activation='relu', kernel_constraint=MaxNorm(3)))
model1.add(MaxPooling2D(pool_size = (2,2), strides=2))
model1.add(Dropout(0.5))
model1.add(Conv2D(128,(3,3), activation='relu', kernel_constraint=MaxNorm(3)))
model1.add(MaxPooling2D(pool_size = (2,2), strides=2))
model1.add(Dropout(0.5))
model1.add(Conv2D(128,(3,3), activation='relu', kernel_constraint=MaxNorm(3)))
model1.add(Conv2D(128,(3,3), activation='relu', kernel_constraint=MaxNorm(3)))
model1.add(MaxPooling2D(pool_size = (2,2), strides=2))
model1.add(Dropout(0.5))
model1.add(Flatten())
model1.add(Dense(512, activation='relu', kernel_constraint=MaxNorm(3)))
model1.add(Dropout(0.5))
model1.add(Dense(512, activation='relu', kernel_constraint=MaxNorm(3)))
model1.add(Dropout(0.5))
model1.add(Dense(10, activation='softmax', kernel_constraint=MaxNorm(3)))

model1.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model1.summary()
```

First the needed Keras modules are imported, and then the model is defined as *Sequential*. Once the model is declared, all the layers are added. The parameter $kernel_constraint = MaxNorm(3)$ will limit the norm of the weights to not exceed the value given and thus prevent over fitting. The loss function used for this model is cross-entropy.

After the model is created, it is time to train it with the following self-defined function:

```
def trainModel(model, epochs):
    return model.fit(xTrain, yTrainOneHot, epochs=epochs, verbose=1,
                     validation_data=(xTest,yTestOneHot))
```

The object returned by this function is a History type object that allows for plotting with this code:

```
fig, axs = plt.subplots()
# summarize history for accuracy
axs.plot(history1.history['accuracy'])
axs.plot(history1.history['val_accuracy'])
axs.plot(history2.history['accuracy'])
axs.plot(history2.history['val_accuracy'])
axs.set_title('Model Accuracy')
axs.set_ylabel('Accuracy')
axs.set_xlabel('Epoch')
axs.legend(['train lr=0.001', 'validate lr=0.001', 'train lr=0.002', 'validate lr=0.002'], loc='l


fig, axs = plt.subplots()
axs.plot(history1.history['loss'])
axs.plot(history1.history['val_loss'])
axs.plot(history2.history['loss'])
axs.plot(history2.history['val_loss'])
axs.set_title('Model Loss')
axs.set_ylabel('Loss')
axs.set_xlabel('Epoch')
axs.legend(['train lr=0.001', 'validate lr=0.001', 'train lr=0.002', 'validate lr=0.002'], loc='u
```

4 Simulation

For simulation, two different models were used. Both of them used the same architecture; however, the first model used the Adam optimizer with the default parameters (learning rate of 0.001) and dropout of 0.5 and the second model used a learning rate of 0.002 and a dropout of 0.2. Both models were train over 100 epochs, which lasted a bit over 5 hours each.

As it can be observed in the Figures 2 and 3, the second model performs better overall in terms of accuracy and loss, roughly reaching 72% accuracy on both training an validation sets.

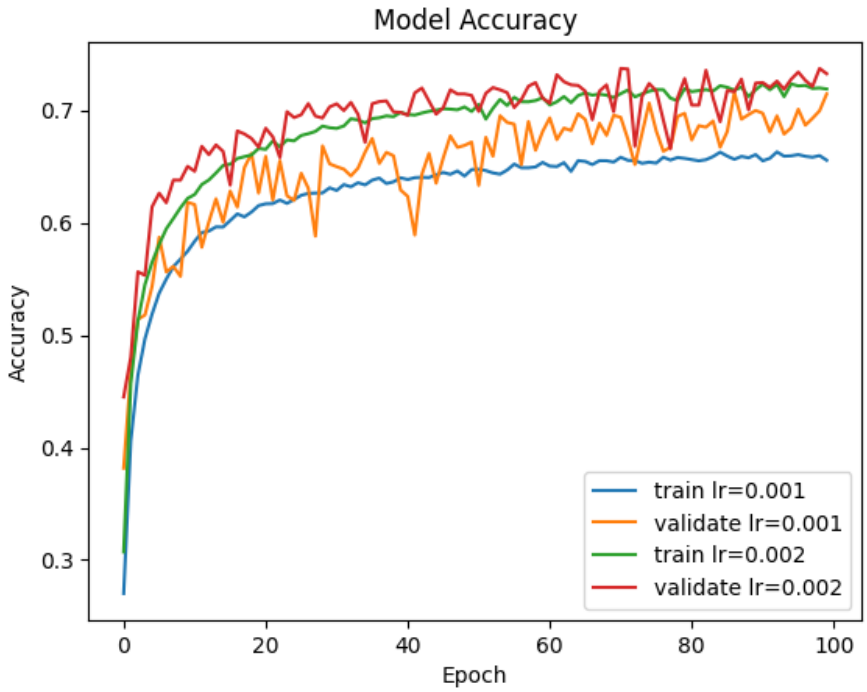

Figure 2: Accuracy of both models

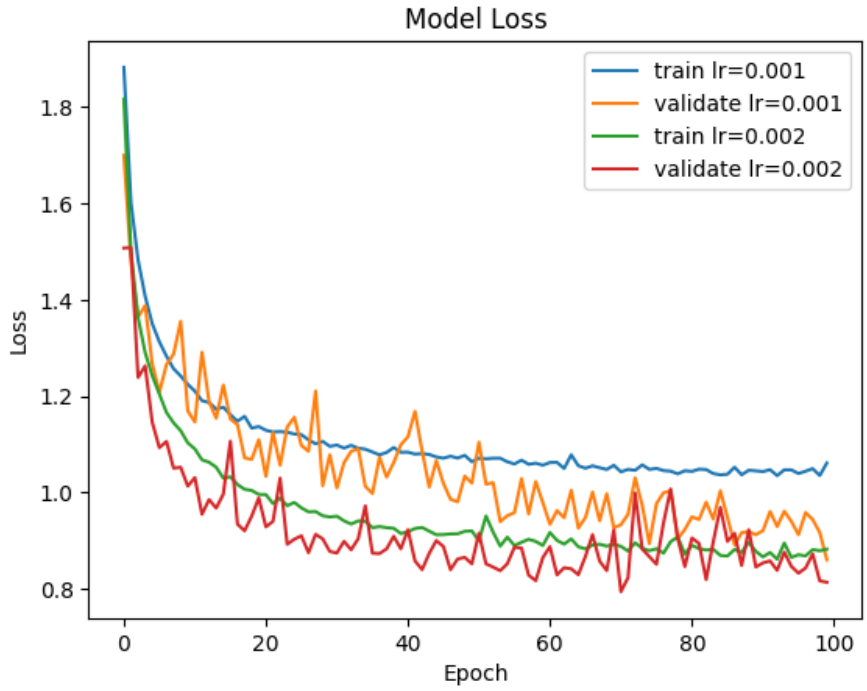

Figure 3: Loss of both models

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems* (F. Pereira, C. Burges, L. Bottou, and K. Weinberger, eds.), vol. 25, Curran Associates, Inc., 2012.

[2] A. Akwaboah, "Convolutional neural network for cifar-10 dataset image classification," 11 2019.