Source:

#ref #ret

---

1 | **Opt 3**

Looked at the Convolutional Neural Network Notebook - What type of data are they using? - They are using images as their input data. - What conversions (if any) had to be done to the data before it could be put into the neural network? - For the basic neural network, they reshape the data to be individual vectors, make them float32, normalize the data, then convert the vectors to binary class matrices via one hot encoding. For the CNN, they repeat the previous steps except without reshaping the data into unrolled input vectors. - What is the output of the neural network, both in terms of what it looks like to the computer (e.g. integers in the range 0-2) and how humans should interpret it (e.g. the type of iris)? - The output should be an array of probabilities for each category, which can be interpreted as, at a given index in the array, the item's probabilities for belonging in each category. - How many hidden layers does the network have, and what type are they (e.g. fully connected, convolutional, recurrent, LSTM, sparse, etc.)? - For the final iteration of the CNN, the model has four hidden layers — two convolutional, and two dense. - What activation function(s) does it use? - The CNN used ReLU and softmax. - What loss or cost function is it using? - The model's loss function is categorical crossentropy - What kind of validation (if any) are they using? - The model uses accuracy as it's validation metric. - What other validation methods might work for this type of problem? - Any validation that works for classification problems, such as recall, f-measure, or precision. - Why do you think the authors may have chosen this architecture for their network? - They started with a small network to help illustrate the concepts more clearly, then gradually added more layers. They used a CNN so the model would be able to perform feature based recognition. - Are there any changes you might try, if you were going to improve on their model? - I will try to add some more layers, as well as some more dropout.

2 | **Opt 3 pt. 2!**

2.1 | **Iteration One: More Regularization**

Tried to add more regularization with this model setup:

```
model = Sequential()
model.add(Conv2D(128, (3, 3), padding='same', input_shape=(32,32,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Dropout(0.25))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(256))
model.add(Activation('relu'))
model.add(Dropout(0.25))
model.add(Dense(100))
```

```
model.add(Activation('relu'))
model.add(Dropout(0.25))
model.add(Dense(num_classes))
model.add(Activation('softmax'))
model.summary()
```

After *hours* of training, the model produced these results:

```
Test loss: 0.930192768573761
Test accuracy: 0.7660999894142151
Training loss: 0.057676762342453
Training accuracy: 0.9868000149726868
```

Which is roughly *one percent worse.*

great.

2.2 | **I2: Better Placed Regularization**

Once realizing I wasn't on a GPU, and trying out this new model setup:

```
model = Sequential()
model.add(Conv2D(128, (3, 3), padding='same', input_shape=(32,32,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25)) ## CHANGED ##
model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(256))
model.add(Activation('relu'))
model.add(Dropout(0.25)) ## CHANGED ##
model.add(Dense(100))
model.add(Activation('relu'))
model.add(Dropout(0.25)) ## CHANGED ##
model.add(Dense(num_classes))
model.add(Activation('softmax'))
model.summary()
```

It produced these results:

```
Test loss: 0.7443265914916992
Test accuracy: 0.7879999876022339
Training loss: 0.08786039799451828
Training accuracy: 0.9797599911689758
```

Which, when compared with the original results:

```
Test loss: 0.9660877988576889
Test accuracy: 0.779
Training loss: 0.024635728995651005
Training accuracy: 0.99504
```

Is better!

2.3 | **I3: More of That**

Dialed up dropout again…

```
model = Sequential()
model.add(Conv2D(128, (3, 3), padding='same', input_shape=(32,32,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.35)) ## CHANGED ##
model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(256))
model.add(Activation('relu'))
model.add(Dropout(0.35)) ## CHANGED ##
model.add(Dense(100))
model.add(Activation('relu'))
model.add(Dropout(0.25))
model.add(Dense(num_classes))
model.add(Activation('softmax'))
model.summary()
```

```
Test loss: 0.6344813704490662
Test accuracy: 0.7918000221252441
Training loss: 0.20304201543331146
Training accuracy: 0.9369400143623352
```

Which gave an even better score!

2.4 | **I4: More Layers**

```
model = Sequential()
model.add(Conv2D(128, (3, 3), padding='same', input_shape=(32,32,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
```

```
model.add(Dropout(0.25)) ## CHANGED ##
model.add(Conv2D(128, (3, 3), padding='same', input_shape=(32,32,3))) ## CHANGED ##
model.add(Activation('relu')) ## CHANGED ##
model.add(MaxPooling2D(pool_size=(2, 2))) ## CHANGED ##

model.add(Dropout(0.35))
model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.35))
model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(256))
model.add(Activation('relu'))
model.add(Dropout(0.35))
model.add(Dense(100))
model.add(Activation('relu'))
model.add(Dropout(0.25))
model.add(Dense(num_classes))
model.add(Activation('softmax'))
model.summary()

Test loss: 0.5848628878593445
Test accuracy: 0.8004999756813049
Training loss: 0.34681835770606995
Training accuracy: 0.878600001335144
```

Our test accuracy and training accuracy are approaching each other.

2.5 | **I(9?): More More Layers**

```
model = Sequential()
model.add(Conv2D(128, (3, 3), padding='same', input_shape=(32,32,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Dropout(0.25))
model.add(Conv2D(128, (3, 3), padding='same', input_shape=(32,32,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Dropout(0.35)) ## CHANGED ##
model.add(Conv2D(128, (3, 3), padding='same', input_shape=(32,32,3))) ## CHANGED ##
model.add(Activation('relu')) ## CHANGED ##
model.add(MaxPooling2D(pool_size=(2, 2))) ## CHANGED ##

model.add(Dropout(0.35))
model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
```

```
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.35))
model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(256))
model.add(Activation('relu'))
model.add(Dropout(0.35))

model.add(Dense(256)) ## CHANGED ##
model.add(Activation('relu')) ## CHANGED ##
model.add(Dropout(0.35)) ## CHANGED ##

model.add(Dense(100))
model.add(Activation('relu'))
model.add(Dropout(0.25))
model.add(Dense(num_classes))
model.add(Activation('softmax'))
model.summary()

Test loss: 0.5869606137275696
Test accuracy: 0.8033999800682068
Training loss: 0.4103561043739319
Training accuracy: 0.8682600259780884
```

After quite some more experimenting, these were around the best results I could get. I achieved this by adding two new layers, and a significant amount more dropout from the original model. However, this process was mostly guess and check. I assume that as I get more practice with this kind of work and gain a deeper understanding, I will get better at iterating more quickly and more effectively. Right now, however, I don't know how those skills would manifest.