Machine Learning Summative

wbbz74

for 22nd March 2019

1 Image Classifier

1.1 Linear Network

The base network consists of a simple, 3 layer fully connected network containing only 1 hidden layer. This performs poorly as there is little opportunity for the network to learn deep and abstract representations by using a single hidden layer. Additionally, spatially related information is not considered together—a vital aspect of an image classification task, where neighbouring pixels need to be considered as a unit in order to identify items in given images.

It was quickly established that the greater number of epochs, the greater overall accuracy of the trained network. After around 50 epochs, this network only shows accuracy of around 15%, also showing signs of overfitting as the training data starts to obtain higher recall accurately than the test data.

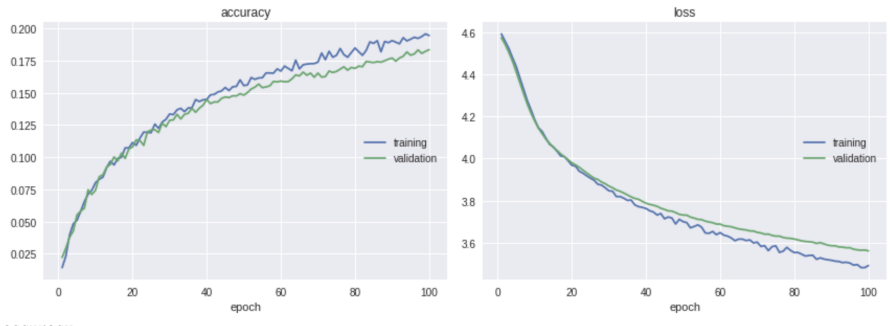

Figure 1: Initial accuracy, 3 fully connected layers (1 hidden) with ReLU normalisation.

1.2 Convolutions

To remedy this, we replace the pure linear structure of the network with a convolution based network. Convolutional Neural Networks (CNNs) are able to better learn representations based on a number of spatially related inputs, allowing for better generalisation of image data. This is achieved by learning convolution masks and which reduce the dimensionality of the image.

My first modified network consisted of 2 convolutional layers, each followed by a max pooling layer (2x2). Then, flattening this output, it is passed to two hidden fully-connected layers before the final classification layer. As shown in Figure 6, after 100 epochs the network shows far fewer signs of overfitting as the accuracy climbs past 20%. However, the accuracy did not climb as quickly as I expected it to compared to the prior network.

```
class BetterNetwork(nn.Module):

    def __init__(self):
        super(BetterNetwork, self).__init__()
        conv = nn.ModuleList()
        conv.append(nn.Conv2d(3,6,5))
        conv.append(nn.MaxPool2d(2,2))
        lin.append(nn.ReLU())
        conv.append(nn.Conv2d(6,12,3))
        conv.append(nn.MaxPool2d(2,2))
        lin.append(nn.ReLU())
        self.conv = conv
        lin = nn.ModuleList()
        lin.append(nn.Linear(in_features=432, out_features=512))
        lin.append(nn.ReLU())
        lin.append(nn.Linear(in_features=512, out_features=100))
        self.lin = lin

    def forward(self, x):
        for l in self.conv:
            x = l(x)
        x = x.view(x.size(0), -1) # flatten
        for l in self.lin:
            x = l(x)
        return x
```

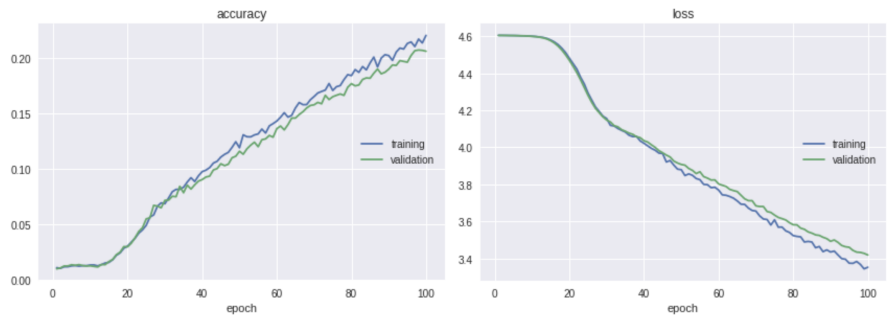

Figure 2: Initial convolution based network (2 conv2d layers, 2 maxpooling, 2 fully-connected layers.

1.3 Regularisation & Optimiser

To reduce the signs of this overfitting and to train a more accurate network, I added dropout layers after each max pooling and the first fully connected layer, each with probability 0.1. We clearly notice far similar performance for training and validation, such that allowing the training to run for around 180 epochs yields a far better network, with overfitting starting to occur much later.

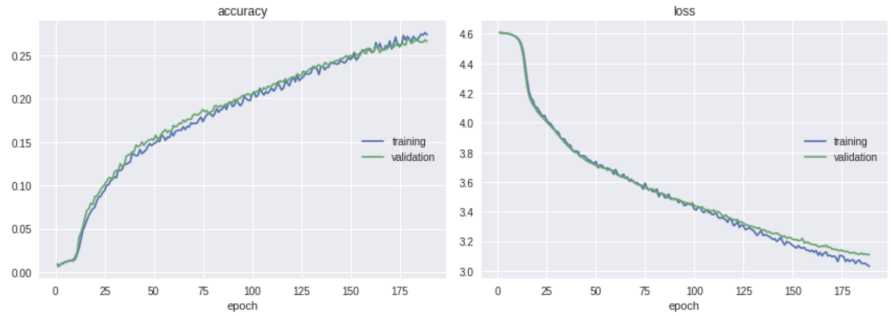

Figure 3: Dropout layers (p=0.2) after each network segment.

Changing this dropout later to 0.2, I noticed marginal improvements to this point where the overtraining occurs.

The most remarkable improvement occurred when changing the optimiser to *Adam*. The network learned far more quickly, but also began overfitting far more quickly after only around 10 epochs. This, however, is still a vast improvement.

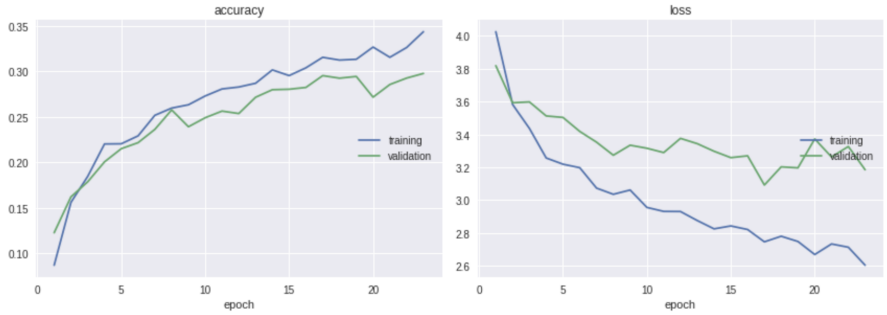

Figure 4: Adam optimiser. Learns—and overfits—far more quickly.

This overfitting was able to be slightly reduced by making improvements to the core network. Namely, using smaller convolution kernels of `3x3` for each convolutional layer and reducing the output channel sizes to 5 and 8. A small improvement is able to be observed.

Figure 5: After modification—slightly reduced overfitting.

I found that augmenting the training images with random rotations, saturation changes and flips reduced the effect of the overfitting on the network and was able to achieve roughly the same peak accuracy on the test set of around 30%, but took far longer to train (25 vs. 115 epochs until overfitting). In further tests, I found it prevented optimal performance, so I disabled this.

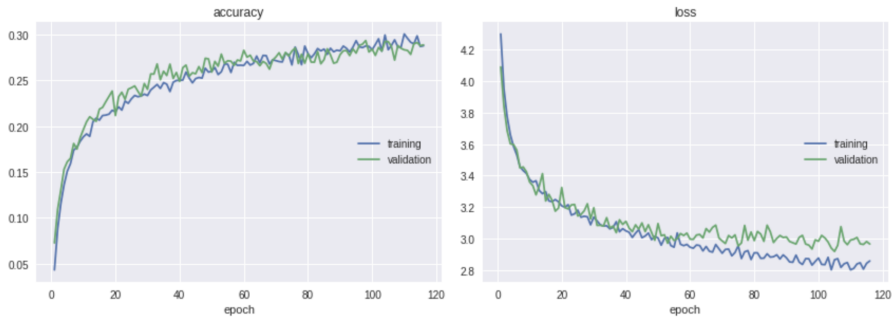

Figure 6: Far longer training with image augmentation, obtaining similar peak accuracy of around 30%.

Additionally, I found that batch normalisation on the output of the convolutional layers was able to further increase the amount of time before over-fitting occurs and offered a slight increase in the accuracy of the network.

1.4 Further Network Improvements

After experimenting with tweaks to the dimensions of the penultimate linear layer I found that size had little effect (unless made to be far too small (< 30) or far too large (> 2048) where performance slightly reduced) on network performance. Additionally, I found adding more than 2 hidden linear layers resulted in reduced network accuracy after a similar number of epochs and the network struggles to improve, also increasing the likelihood of overfitting. This occured for a variety of sizes.

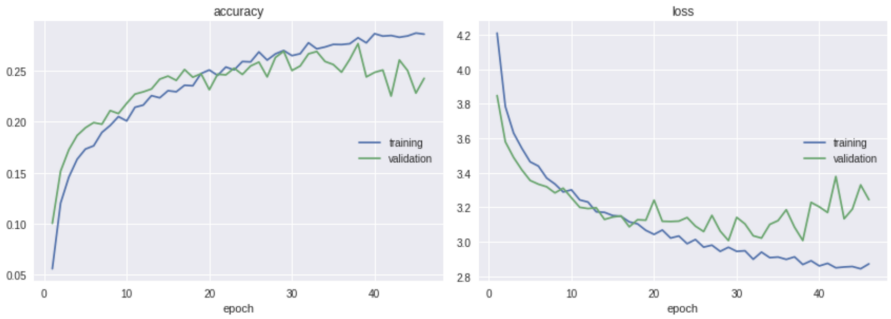

Figure 7: 4 hidden linear layers, greater likelihood of overfitting later on.

Next, I increased the batch size to 32. This should better inform the network at each step how to adjust the particular weights, as more of the dataset is considered at each learning step. I then added another convolution to the network and dramatically increased the number of channels output by each:

```
conv = nn.ModuleList()
conv.append(nn.Conv2d(3,16,3,stride=2))
conv.append(nn.Dropout(0.2))
conv.append(nn.BatchNorm2d(16))
conv.append(nn.ReLU())
conv.append(nn.Conv2d(16,32,3,stride=2))
conv.append(nn.Dropout(0.1))
```

```
conv.append(nn.BatchNorm2d(32))
conv.append(nn.ReLU())
conv.append(nn.Conv2d(32,32,3,stride=2))
conv.append(nn.Dropout(0.1))
conv.append(nn.ReLU())
self.conv = conv
lin = nn.ModuleList()
lin.append(nn.Linear(in_features=288, out_features=512))
lin.append(nn.ReLU())
lin.append(nn.Linear(in_features=512, out_features=100))
self.lin = lin
```

This allowed for much less overfitting and the network to confidently achieve classification accuracy of around 35%.

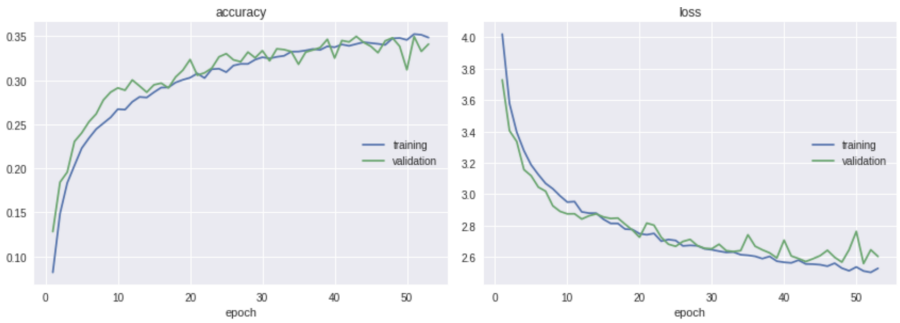

Figure 8: Updated network, featuring much larger output channels from convolutions.

For my final network, I added 3 extra identical convolutions to the end of the network, the same as the final layer, making the convolutional aspect of the network far deeper. This network somewhat started to overfit towards the end but I was unable to find a better performing network that did not overfit. The best test accuracy I was able to obtain is **40.0%**.

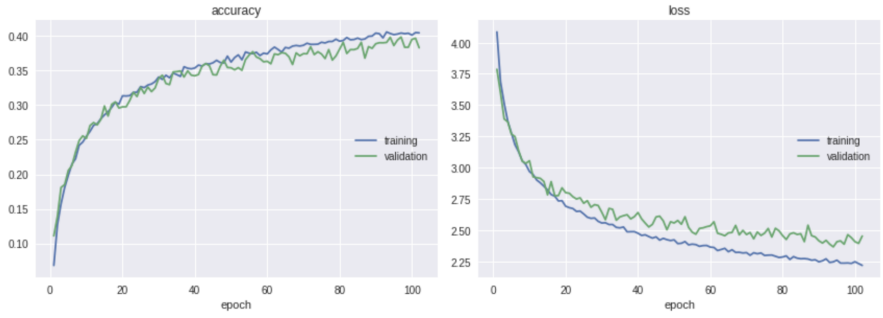

Figure 9: The final network, consiting of 5 convolutional and 2 linear layers.

2 Generating a Pegasus