Abstract

We used the CIFAR-10 dataset to measure the accuracy of CNN and DNN architectures, in order to gain insight to build a future facial recognition model. From the experiments we learned that CNN are better than DNNs for image classification and that regularization is important for CNNs due to their many parameters. Our best model was a 4-layer CNN with dropout and batch normalization used to achieve an accuracy of 78.89%.

Introduction

This week the team has been given the problem of how to go about setting up a facial recognition model, so that mobile users can simply look at their phones rather than having to type in a username and password or a keycode. For this we experiment we used deep learning models with more than one hidden layer and tried using convolutional neural nets (CNN) as well on the image data. For the experimentation, the CIFAR-10 dataset, a balanced data set with 10 classes of various classes like birds and airplanes, with 60000 images.

Literature Review

CIFAR 10 is a famous dataset that has been used for many research papers. One research paper the team reviewed before experimenting was the paper written by Rahul Chauhan, Kamal Ghanshala, and RC Joshi. The team was able to build a model that achieved an accuracy of 80.17% on the test dataset. The model used convolutional neural nets (CNN) and using the regularization techniques of data augmentation and dropout. For the data augmentation the data was mirrored, randomly cropped, color shifted, and rotated. This gave the team some ideas on how to approach the dataset in terms of regularization techniques that could be successful on this dataset. The link to this paper is in the appendix of this paper.

Review research design and modeling methods

The CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class. The initial setup is 50000 images for training and 10000 images for testing. We also separated the training data and set aside 10000 images for validation, leaving 40000 images for testing. All images have labels for the images that correspond with 1 of the following 10 classes: 1. Airplane 2. Automobile 3. Bird 4. Cat 5. Deer 6. Dog 7. Frog 8. Horse 9. Ship 10. Truck. The 10 classes are diverse and a good challenge to experiment with image classification.

For modeling solutions to the CIFAR-10 classification data problem we experimented with deep dense neural nets and convolutional neural nets. Each of our models were deep, meaning that they contained more than one hidden layer. A convolutional neural network learns features by scanning over the image using a window called a kernel that is then summarized during the pooling step causing the features to be down sample and keeping the most prominent ones. For our models we used max pooling techniques which keeps the largest values of the feature map. Typically, the model is then flattened and passed to a dense layer before being passed to an activation function like softmax for classification. An example architecture for a CNN can be seen in the picture below.

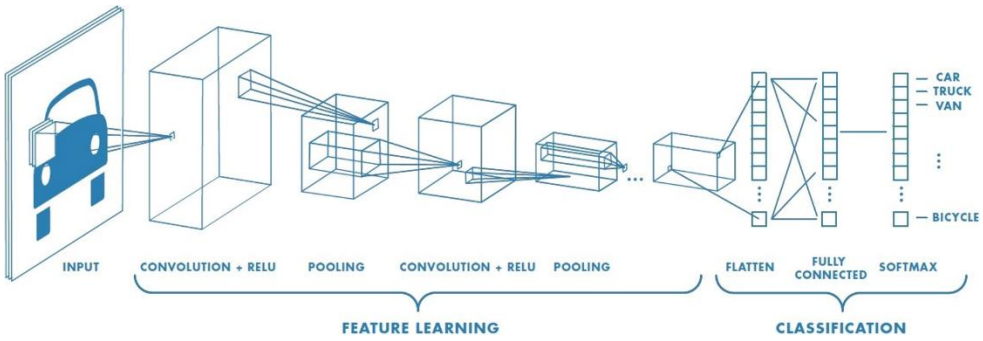

Figure 1: CNN architecture

Implementation and Programming

These models were implemented using the python programming language and several libraries. The main libraries used were pandas for data cleaning and managing, keras tensorflow was used to build the neural networks, sklearn for scaling and confusion matrix, numpy to handle arrays, and matplotlib to build the plots. The sequential class was used to build the neural networks with different activation functions, number of layers, and number of nodes. For regularization techniques we used tensorflow's BatchNormalization and dropout methods to prevent overfitting. The compile method was used with a loss function of SparseCategorialCrossentropy because we did not one hot encode the data. The evaluate method was used to test the accuracy of the model after being fitted on the test data. The development environment was done in jupyter noteboks and the code can be found attached to this report in pdf format.

We approached the problem by conducting 5 experiments. The first experiment we used a deep neural network with 2 layers and no regularization. The second experiment we used a deep neural network with 3 layers and no regularization. For the third experiment we built a CNN with 2 convolution and max pooling layers and no regularization. For the fourth experiment we built a CNN with 3 convolution and max pooling layers and no regularization. For the last experiment we did more experiments by redoing all the models with regularization.

Data Preparation

One of the first steps we had to do to the images was preprocess the data to prepare it to be able to be loaded into our model. We normalized the data by dividing each pixel value by 255. In an image the value 0 represents black and 255 represents white in this image so we divided by 255 to have all numbers be between 0 and 1 for easier training of the model. For the dense deep neural networks we flattened the data for input to an array of 3072 inputs accounting for the 32x32x3 (rgb) image. For the CNN models we did not have to flatten the data before input.

As stated earlier the 10 classes of the dataset were as follows: 1. Airplane 2. Automobile 3. Bird 4. Cat 5. Deer 6. Dog 7. Frog 8. Horse 9. Ship 10. Truck. Below is sample data for a few of the classes. As you can see, some of the images are tough even for a person to classify, like the first image which is a bird.

Figure 2: Example images of CIFAR-10 data

Results

In this section we will go over the results of the neural net models. We first reviewed DNNs, then CNNs, then both with regularization like dropout and batch normalization. Some of the constants between models is an optimizer of adam, loss function of SparseCategoricalCrossentropy, 20 epochs, and batch size of 512.

Experiment 1

For the first experiment we analyzed a two layer DNN with the first layer containing 256 hidden nodes and an activation layer of relu, then a second layer of 512 hidden nodes with an activation layer of relu, and finally an output layer of 10 nodes, one for each class, with an activation layer of softmax. A picture of the architecture can be seen below.

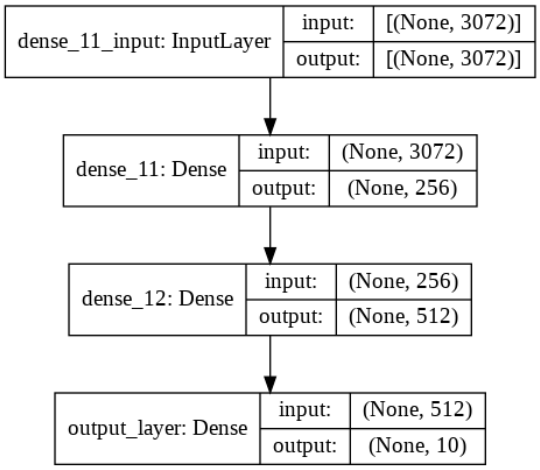

Figure 3: Experiment 1 architecture

With this architecture the model achieved an accuracy of 51.83%

Experiment 2

For the second experiment we analyzed a three hidden layer DNN with the first layer containing 256 hidden nodes and an activation layer of relu, then a second layer of 512 hidden nodes with an activation layer of relu, a third layer with 1024 hidden nodes and an activation layer of relu, and finally an output layer of 10 nodes, one for each class, with an activation layer of softmax. A summary of the architecture can be seen below.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_2 (Dense)              (None, 256)               786688
_________________________________________________________________
dense_3 (Dense)              (None, 512)               131584
_________________________________________________________________
dense_4 (Dense)              (None, 1024)              525312
_________________________________________________________________
output_layer (Dense)         (None, 10)                10250
=================================================================
Total params: 1,453,834
Trainable params: 1,453,834
Non-trainable params: 0
```

Figure 4: Experiment 2 model summary

With this architecture the model achieved an accuracy of 50.62%.

Experiment 3

For the third experiment we trained the model using a CNN. This model was 2 convolutional layers with 2 max pooling layers as well. The first layer used 256 features with a kernel size of 3x3 and stride of 1, the second layer was max pooling layer of 2x2 and used a stride of 2, the third layer used 512 features with a kernel size of 3x3 and stride of 1, the fourth layer was a max pooling layer of 2x2 and used a stride of 2, the next layer was a flatten layer that then passed the output to a dense 512 hidden node layer with an activation function of relu, and finally an output layer of 10 nodes, one for each class, with an activation layer of softmax. These CNN models took much longer than the DNN models with the CNN model for experiment 3 taking 1987 seconds. This model achieved an accuracy of 71.78% on the test dataset, which is much better than the DNN models.

Experiment 4

For the fourth experiment we trained the model using a CNN. This model was 3 convolutional layers with 3 max pooling layers as well. The first layer used 256 features with a kernel size of 3x3 and stride of 1, the second layer was max pooling layer of 2x2 and used a stride of 2, the third layer used 512 features with a kernel size of 3x3 and stride of 1, the fourth layer was a max pooling layer of 2x2 and used a stride of 2, the fifth layer was another convolutional layer with 1024 features with a kernel size of 3x3 and stride of 1, the sixth layer was a max pooling layer of 2x2 and used a stride of 2, the next layer was a flatten layer that then passed the output to a dense 512 hidden node layer with an activation function of relu, and finally an output layer of 10 nodes, one for each class, with an activation layer of softmax. A

summary can be seen in the figure below and it should be noted the high number of parameters the model trains on when CNNs are used. With CNNs our parameters are in the millions, while only about a million when using dense deep neural nets. This model did slightly better than the last, achieving an accuracy score of 73.43 on the test dataset.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 30, 30, 256)       7168
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 15, 15, 256)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 13, 13, 512)       1180160
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 6, 6, 512)         0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 4, 4, 1024)        4719616
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 2, 2, 1024)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 4096)              0
_________________________________________________________________
dense_9 (Dense)              (None, 512)               2097664
_________________________________________________________________
dense_10 (Dense)             (None, 10)                5130
=================================================================
Total params: 8,009,738
Trainable params: 8,009,738
Non-trainable params: 0
```

Figure 5: Experiment 4 model summary

When looking at the loss and validation graph for the model on the training and validation data we can see that there the model was overfitting. In the figure below we can see by the divergence that the model is overfitting. To improve results, we need to introduce some regularization into the model.

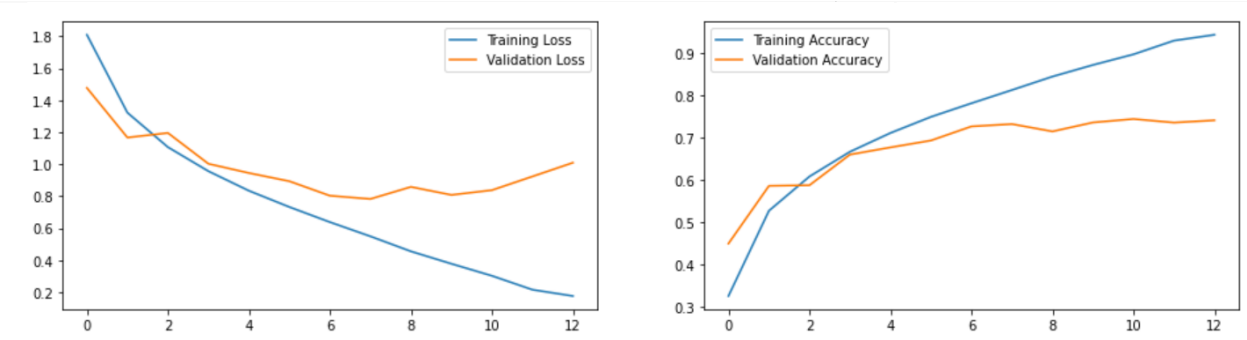

Figure 6: Loss and accuracy of experiment 4 model

Experiment 5

After the first four experiments we can see that the models were overfitting. The CNN models overfit the most due to the many parameters of the models. We redid the previous experiment using regularization. The first form of regularization we performed was adding a dropout layer for every

hidden layer. Each dropout layer had a value of 20%. This improved results with a table of the accuracies below.

The next regularization technique we used was batch normalization. A batch normalization layer was added then the models were retested using the same architecture with a batch normalization layer. The accuracies for the normalization and dropout can be seen in the table below.

Table 1: Experiments using regularization

Layers	Regularization	Neural Net Type	Proc. Time (seconds)	Train Acc. (%)	Val Acc. (%)	Test Acc. (%)
2	Dropout	DNN	20	46.65	46.80	46.62
3	Dropout	DNN	25	50.54	48.93	47.83
2	Dropout	CNN	2300	86.42	73.85	73.01
3	Dropout	CNN	3800	87.89	74.61	74.07
2	Batch Normalization	DNN	21	74.95	37.30	37.65
3	Batch Normalization	DNN	50	89.04	44.87	44.77
2	Batch Normalization	CNN	1980	100	74.97	74.58
3	Batch Normalization	CNN	3400	95.34	70.79	70.67

From this we were able to see that dropout and batch normalization had similar positive effects on the accuracy of our model. We then combined dropout and batch normalization using the architecture of experiment 4 to achieve an accuracy of 76.66% on the test data set.

The last model we test was our most accurate model on the test data set. For the last model we added one more Convolutional layer of 256 features to the architecture used in experiment 4. We also used the regularization techniques of batch normalization and dropout. The summary of the model can be seen below.

Layer (type)	Output Shape	Param #
conv2d_65 (Conv2D)	(None, 30, 30, 256)	7168
conv2d_66 (Conv2D)	(None, 28, 28, 256)	590080
max_pooling2d_56 (MaxPooling	(None, 14, 14, 256)	0
dropout_26 (Dropout)	(None, 14, 14, 256)	0
conv2d_67 (Conv2D)	(None, 12, 12, 512)	1180160
max_pooling2d_57 (MaxPooling	(None, 6, 6, 512)	0
conv2d_68 (Conv2D)	(None, 4, 4, 1024)	4719616
batch_normalization_7 (Batch	(None, 4, 4, 1024)	4096
max_pooling2d_58 (MaxPooling	(None, 2, 2, 1024)	0
dropout_27 (Dropout)	(None, 2, 2, 1024)	0
flatten_12 (Flatten)	(None, 4096)	0
dense_24 (Dense)	(None, 512)	2097664
dense_25 (Dense)	(None, 10)	5130

Total params: 8,603,914
Trainable params: 8,601,866
Non-trainable params: 2,048

Figure 7: Model summary of best model

With this architecture we achieved our best accuracy on the test data at 78.85%. In the figure below we can see the confusion matrix for this model and we can see the balanced predictions as indicated by the diagonal line of dark blue squares.

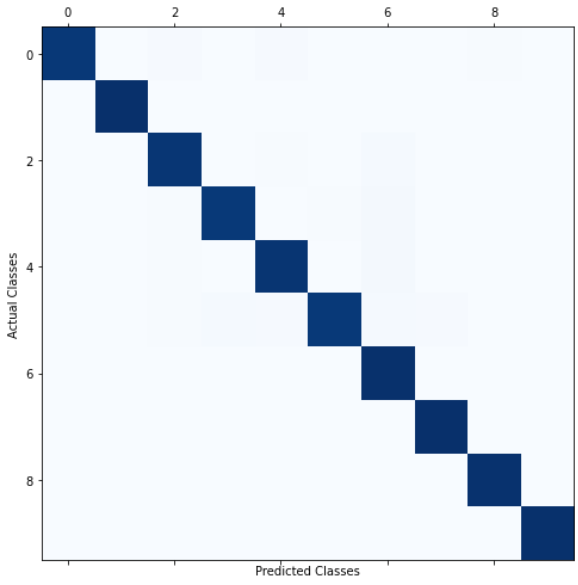

Figure 8: Confusion Matrix of best model

When looking at the predictions of the best model we looked at the predictions of the model for the first 20 pictures in the training set. From the image below you can see that the model was confident in most of the images, with a prediction score in the 90s, but there were some images like the first one where there was some uncertainty.

	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
0	0.00%	0.00%	0.01%	59.62%	0.01%	23.13%	17.22%	0.00%	0.00%	0.00%
1	0.00%	0.59%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	99.41%	0.00%
2	0.05%	59.27%	0.00%	0.02%	0.00%	0.00%	0.00%	0.00%	40.60%	0.07%
3	99.71%	0.00%	0.01%	0.00%	0.01%	0.00%	0.00%	0.00%	0.27%	0.00%
4	0.00%	0.00%	0.00%	0.01%	0.08%	0.00%	99.91%	0.00%	0.00%	0.00%
5	0.00%	0.00%	0.00%	0.00%	0.00%	0.11%	99.89%	0.00%	0.00%	0.00%
6	0.00%	97.71%	0.00%	0.00%	0.00%	0.01%	0.00%	0.00%	0.00%	2.27%
7	0.00%	0.00%	0.02%	0.00%	0.01%	0.00%	99.97%	0.00%	0.00%	0.00%
8	0.00%	0.00%	0.00%	99.99%	0.00%	0.00%	0.01%	0.00%	0.00%	0.00%
9	0.06%	39.93%	0.01%	0.02%	0.31%	0.02%	1.90%	0.00%	0.01%	57.74%
10	65.63%	0.00%	1.78%	2.18%	28.60%	0.32%	0.01%	1.20%	0.28%	0.00%
11	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	100.00%
12	0.00%	0.00%	0.01%	3.53%	0.33%	96.02%	0.07%	0.04%	0.00%	0.00%
13	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	100.00%	0.00%	0.00%
14	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	100.00%
15	0.13%	0.03%	0.01%	0.31%	0.33%	0.00%	60.30%	0.00%	38.89%	0.00%
16	0.00%	0.00%	0.00%	0.70%	0.00%	99.28%	0.00%	0.01%	0.00%	0.00%
17	0.00%	0.00%	0.00%	8.13%	3.76%	4.32%	2.85%	80.63%	0.00%	0.31%
18	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	100.00%	0.00%
19	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	100.00%	0.00%	0.00%	0.00%

Figure 9: Predictions of first 20 images from training set for the best model

When looking closer at the first image we see that it was split between the image being a cat, dog, or frog. By looking at the image below we can see that the image was indeed a cat, but can understand the uncertainty as the defining features of a cat are blurred a bit in this picture.

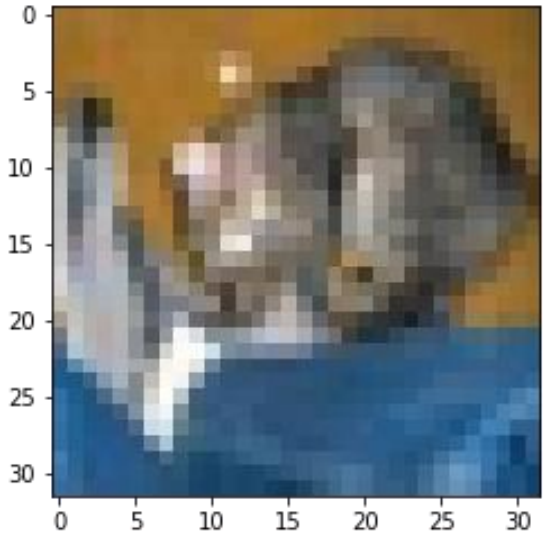

Figure 10: First image in training set

From the experiments we learned that regularization is important for CNN models because they can easily overfit due to the many parameters. The regularization techniques did not help the accuracy of the DNN models in this experiment. In the table below is the result of all our models. The best model is highlighted.

Table 2: Results from all experiments

Layers	Regularization	Neural Net Type	Proc. Time (seconds)	Train Acc. (%)	Val Acc. (%)	Test Acc. (%)
2	None	DNN	20	61.69	52.42	51.83
3	None	DNN	25	59.42	48.52	50.62
2	None	CNN	1960	94.10	71.99	71.78
4	None	CNN	3200	94.63	74.09	73.42
2	Dropout	DNN	20	46.65	46.80	46.62
3	Dropout	DNN	25	50.54	48.93	47.83
2	Dropout	CNN	2300	86.42	73.85	73.01
3	Dropout	CNN	3800	87.89	74.61	74.07
2	Batch Normalization	DNN	21	74.95	37.30	37.65
3	Batch Normalization	DNN	50	89.04	44.87	44.77
2	Batch Normalization	CNN	1980	100	74.97	74.58
3	Batch Normalization	CNN	3400	95.34	70.79	70.67
3	Dropout/Nomalization	CNN	3640	96.11	76.46	76.66
4	Dropout/Normalization	CNN	7400	88.85	79.15	78.89

Conclusion

In conclusion the team was able to gain some very important insight into the workings of deep neural networks and convolutional neural networks. Our best model was a CNN that achieved an accuracy of 78.89% on the test data set and used the regularization techniques dropout and normalization. From the experiments we measured that CNNs were better than DNNs for classifying the image data. We also saw that CNNs can easily overfit due to the many parameters. This led us to introduce some regularization techniques into the models. We tested dropout layers of 20% and batch normalization for the models. These improved the accuracy of the CNN models and hurt the accuracy of the DNN models. When developing a model to classify a person's face for unlocking their phone we will need to introduce regularization to the model to help it from overfitting, so if a person is wearing sunglasses for example that the model can still accurately classify the person. From this experiment we suggest moving forward using a CNN with regularization as the approach for building a facial recognition model.

Appendix

https://ieeexplore.ieee.org/abstract/document/8703316?casa_token=YKhInnACtqkAAAAA:t6oq6b0vuPcMtfhCHXs4Uh5Ha3CztgQz6n04v5vPuIVrJxWK4Dsc--IvfsXAr_KDPhfmOMcjYw