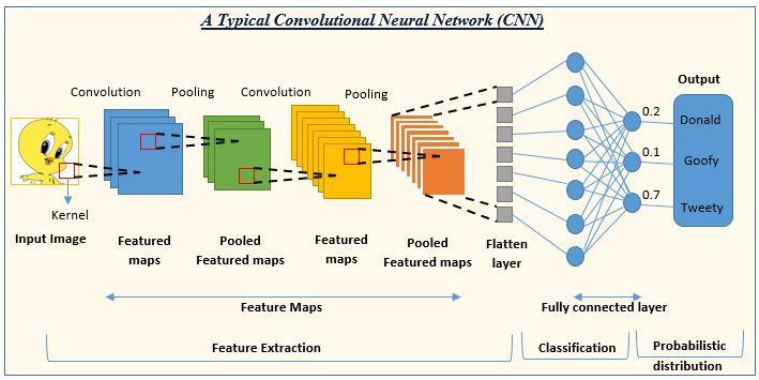
Individual Project Report
Kiyan Mohebbizadeh
UNI: km3826

CNN:

The idea behind a convolutional neural network (CNN) is a machine learning model that can take an image and classify it based on learned patterns. It does so by first processing the image in convolutional layers to extract the most important features from the image, then passing those features through a neural network that then finds the patters and relates them to the targets to determine the classification of an image.



In the above diagram representing a standard CNN we can see that there are two convolutional layers. Their jobs are to run a kernel over the image and extract features from the image. These features can be edges, colors, gradients, etc. For convolutional layers we can decide whether they use padding, the stride length for the kernel, the kernel size, the activation function, the number of features in the output, etc.

We also see two pooling layers in the above CNN. These layers equate to a type of dimensionality reduction. They work to reduce the spatial dimensions of an image. They pass a kernel over an image and either take the maximum value or the average value (the two most common pooling layers) and from that compose a new smaller image to be passed to the following layer.

After the feature extraction has been completed. We need to flatten the feature maps into a one-dimensional vector and pass them into the neural network. Flattening essentially rearranges the image into a vector for use in the following layers.

Once the image has been converted to a vector, we can then use a traditional neural network with connected layers to determine parameters and classify the image in the final layer. In these layers we can set the number of nodes and the activation functions for the nodes. The model then determines the highest likelihood class for the image and outputs that as the predicted class.

Neural Networks do struggle with overfitting at times so there are several ways to combat this. First, we can use dropout layers as well as batch normalization layers. Drop out randomly blocks some nodes in training of the model to reduce the likelihood that the model is too dependent of any node. Batch normalization works by normalizing the batches between layers to again reduce the likelihood of over-reliance on a particular feature. We can also use image augmentation to slightly change the training set on each epoch not allowing the model to over rely on a particular set of images, their spacings, orientations etc. And finally, we can reduce the batch size of the image (at a tradeoff for efficiency) but by training in small batches, we reduce the likelihood of sticking to the large features of a training data set.

To put it as simply as possible, a CNN is better for image classification because it gather data from the processing of the raw image as well as from the actual analysis of the image in vector form. Having these functions combined allows from high level analysis and classification of images.

Code Logic and Parameters:

The parameters are described in detail in the code; however, I will describe general parameter selection factors. Let's start by looking at the data I are provided. I know that I have a very clean data set. The images are all cropped around the street signs, the sizes of the images are all the same, and everything about the data is relatively clean.

Knowing that the data is very clean, I can keep a relatively small validation set so I split the training by a 90:10 split. However, I also know that with small data sets it is more likely to leave out one category or another, so I stratify the split so that the validation set tests the model on all the categories. A small validation set is also better because it allows there to be more data left for training the model.

The next set of parameters are the ones in the CNN layers. Let's start with the convolutional layers. I adopted these layers from LeNet5. Padding is only applied in the first layer, the following convolutional layers do have any padding. I want our stride to be 1 to gather as much information as possible from the image, and 6, 16, 120 is a standard filter rate for CNNs. In this model I reduced the kernel size in the convolutional layers because the images are very clean, so there is not much noise to filter out. The smaller kernel size allows for more data to be passed from one layer to the next. I also opted to adjust the activation function from the standard "relu" to "elu" from which I experienced better results. The activation function is stands for Exponential Linear Unit and seemed to work the best in modeling this data.

I decided to use average pooling since there was not much noise in the data. For noisier data Max pooling works better, however, max pooling can also remove features from the data as well. For this reason, I use average pooling with a small pool size to not reduce dimensionality too much.

For the Dense Layers I used a standard narrowing structure where features are narrowed down until a decision is made. 256, 128, 64, 32 and 16 are standard number of nodes for dense layers. Again, I opted for the “elu” activation function for these layers. The last dense layer is different however, the soft max function selects the node with the highest weight (highest probability) and there are the same number of nodes as categories, so each node in the last Dense layer represents a category.

Finally, there are 3 dropout Layers. I set the dropout rate relatively low as to not impede in the actual training of the model, but these layers are just helping ensure that the large Dense layers don’t over fit to any feature.

In compiling the model, we use categorical cross-entropy since we encoded our target variable. And since this project is based around the accuracy of our model, we use accuracy as a metric to evaluate the model at each epoch. For the optimizer function I found the Adamax worked best with this model and data set. The role of the optimizer function is just selecting a formula for which the weights of certain parameters and learning rates are set.

The Parameters for the image augmentation function were set to create a “jitter” effect in the data. Since the images are so clean and cropped, setting these values larger could result in cutting out the target of the image. And these images are also rotationally sensitive. If we set the rotation level to high a 6 could be mistaken for a 9. The goal of this part was to remove positional and shape consistency between epochs to reduce the risk of overfitting. For the image augmentation we do need to apply a generator to the validation set for consistency, however, we do not change the validation set at all, so that is why we created two generators and applied them to their respective sets.

I set random seeds for the script to help with consistency, but this does not guarantee consistency across the machines and environments.

Starting Model and Evaluation:

```
cnn = Sequential()
cnn.add(Conv2D(filters=6, kernel_size=(5, 5), activation='relu', strides=(1, 1), padding='same', input_shape=(32, 32, 3)))
cnn.add(MaxPooling2D(pool_size=(2, 2)))
cnn.add(Conv2D(filters=16, kernel_size=(5, 5), activation='relu', strides=(1, 1), padding='valid'))
cnn.add(MaxPooling2D(pool_size=(2, 2)))
cnn.add(Conv2D(filters=120, kernel_size=(5, 5), activation='relu', strides=(1, 1), padding='valid'))
cnn.add(Flatten())
cnn.add(Dense(units=120, activation='relu'))
cnn.add(Dense(units=11, activation='softmax'))

cnn.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
```

I started with a simple LeNet 5 model. With this model I was able to achieve 89% accuracy on the test data set. As you can see, I started with max-pooling layers, a larger

kernel for convolution, different activation, and optimization functions and fewer dense layers.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 6)         456

 max_pooling2d (MaxPooling2D  (None, 16, 16, 6)        0
 )

 conv2d_1 (Conv2D)           (None, 12, 12, 16)        2416

 max_pooling2d_1 (MaxPooling  (None, 6, 6, 16)         0
 2D)

 conv2d_2 (Conv2D)           (None, 2, 2, 120)         48120

 flatten (Flatten)           (None, 480)               0

 dense (Dense)               (None, 120)               57720

 dense_1 (Dense)             (None, 11)                1331

=================================================================
Total params: 110,043
Trainable params: 110,043
Non-trainable params: 0
_________________________________________________________________
```

This original LeNet5 model had 110,043 parameters. Each of these parameters is responsible for keeping track of some factor in the image and later will help in the classification process when the model sees new data.

```
cnn = Sequential()
cnn.add(Conv2D(filters=6, kernel_size=(3, 3), activation='elu', strides=(1, 1), padding='same', input_shape=(32, 32, 3)))
cnn.add(AveragePooling2D(pool_size=(2, 2)))
cnn.add(Conv2D(filters=16, kernel_size=(3, 3), activation='elu', strides=(1, 1), padding='valid'))
cnn.add(AveragePooling2D(pool_size=(2, 2)))
cnn.add(Conv2D(filters=120, kernel_size=(3, 3), activation='elu', strides=(1, 1), padding='valid'))
cnn.add(Flatten())
cnn.add(Dense(units=120, activation='elu'))
cnn.add(Dense(units=11, activation='softmax'))

cnn.compile(optimizer="Adamax", loss="categorical_crossentropy", metrics=["accuracy"])
```

However, we can see that because of these changes the parameters of the model increased and therefore the complexity of the model increased as well. There are now 379899 parameters to consider in the model. However, as we can see from the accuracy measure, we can see that there was also an increase in accuracy.

The next iteration of the model was simply an update to the existing model. In this version of the model, we changed the pooling layers, the activation functions, the optimizer function as well as the kernel sizes for the convolutional layer. With this model I was able to achieve >=92% accuracy on the test set.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 6)         168

 average_pooling2d (AverageP  (None, 16, 16, 6)        0
 ooling2D)

 conv2d_1 (Conv2D)           (None, 14, 14, 16)        880

 average_pooling2d_1 (Averag  (None, 7, 7, 16)         0
 ePooling2D)

 conv2d_2 (Conv2D)           (None, 5, 5, 120)         17400

 flatten (Flatten)           (None, 3000)              0

 dense (Dense)               (None, 120)               360120

 dense_1 (Dense)             (None, 11)                1331

=================================================================
Total params: 379,899
Trainable params: 379,899
Non-trainable params: 0
_________________________________________________________________
```

However, I felt that there was still a way to improve this model, so Instead of having a simple network at the end that just looked at each of the features extracted, I decided to implement a more complete neural network with many layers at the end of the model. This dramatic increase in complexity also required that there were some countermeasures for over fitting which is why I decided to add the drop out layers to the larger dense layers.

```
cnn = Sequential()
cnn.add(Conv2D(filters=6, kernel_size=(3, 3), activation='elu', strides=(1, 1), padding='same', input_shape=(32, 32, 3)))
cnn.add(AveragePooling2D(pool_size=(2, 2)))
cnn.add(Conv2D(filters=16, kernel_size=(3, 3), activation='elu', strides=(1, 1), padding='valid'))
cnn.add(AveragePooling2D(pool_size=(2, 2)))
cnn.add(Conv2D(filters=120, kernel_size=(3, 3), activation='elu', strides=(1, 1), padding='valid'))
cnn.add(Flatten())
cnn.add(Dropout(rate=0.1, seed=42))
cnn.add(Dense(units=256, activation='elu'))
cnn.add(Dropout(rate=0.1, seed=42))
cnn.add(Dense(units=128, activation='elu'))
cnn.add(Dropout(rate=0.1, seed=42))
cnn.add(Dense(units=64, activation='elu'))
cnn.add(Dense(units=32, activation='elu'))
cnn.add(Dense(units=16, activation='elu'))
cnn.add(Dense(units=11, activation='softmax'))

cnn.compile(optimizer="Adamax", loss="categorical_crossentropy", metrics=["accuracy"])
```

As we can see from this model there is now several dense layers following the convolutional step. As well as 3 dropout layers that control for overfitting. This increases the complexity of the model to 830651 parameters considered, but it also allows us to reach >=93% accuracy on the test data.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 6)         168

 average_pooling2d (AverageP  (None, 16, 16, 6)        0
 ooling2D)

 conv2d_1 (Conv2D)           (None, 14, 14, 16)        880

 average_pooling2d_1 (Averag  (None, 7, 7, 16)         0
 ePooling2D)

 conv2d_2 (Conv2D)           (None, 5, 5, 120)         17400

 flatten (Flatten)           (None, 3000)              0

 dropout (Dropout)           (None, 3000)              0

 dense (Dense)               (None, 256)               768256

 dropout_1 (Dropout)         (None, 256)               0

 dense_1 (Dense)             (None, 128)               32896

 dropout_2 (Dropout)         (None, 128)               0

 dense_2 (Dense)             (None, 64)                8256

 dense_3 (Dense)             (None, 32)                2080

 dense_4 (Dense)             (None, 16)                528

 dense_5 (Dense)             (None, 11)                187

=================================================================
Total params: 830,651
Trainable params: 830,651
Non-trainable params: 0
```

Tuning:

There were two forms of tuning that were done to this model. The first is what I like to call “exploratory tuning”. This type of tuning is testing various parameters that have no logical selection process. For example, for the activation functions, you can narrow down the functions by the type of classification, but after that one must test at random to see what works best with your data and model. The same goes for optimization functions, layer sizes, adding or removing layers, etc. The second type of tuning I like to call “logical tuning” this is measured adjustments to the parameters based on the one’s knowledge of the data. Examples of this would be setting the pooling layers to be average pooling instead of max pooling, reducing the kernel size in the convolutional layers, setting the image augmentation function to very low ranges, reducing the batch size in the training of the model, adding dropout layers, etc. Between these two tuning methods, I was able to achieve >=93% accuracy on the test data.

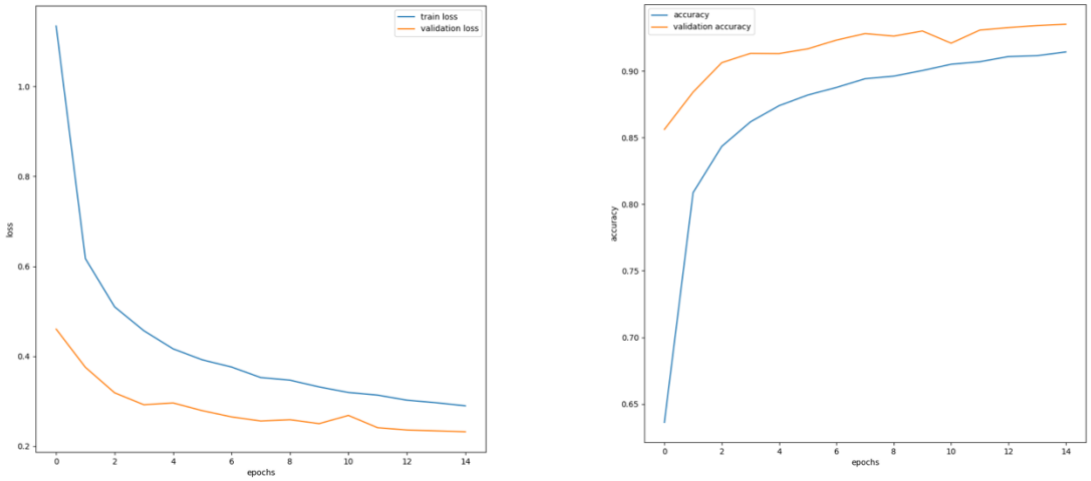
One important factor in tuning was watching the training and validation accuracy at each epoch. If the training accuracy ever surpassed the validation accuracy, I knew there must be some measures taken to reduce overfitting.

Model Evaluation:

To evaluate the model at each epoch I used the validation set, however at the end of the training process I used the test data provided to create the final evaluation and testing of the model. I found that in testing the model the accuracy surpassed 91% by at least 2%.

```
Epoch 15/15
13187/13187 [==============================] - 117s 9ms/step - loss: 0.2988 - accuracy: 0.9115 - val_loss: 0.2324 - val_accuracy: 0.9348
814/814 [==============================] - 5s 6ms/step - loss: 0.2351 - accuracy: 0.9363
Accuracy: 0.9363091588020325

Process finished with exit code 0
```



As we can see in the charts, there is a nonlinear relationship with epochs and the metrics of evaluation. We want to attempt to keep the lines as close to each other as possible because as soon as the training data evaluation improves much more than the evaluation, we can suspect that over fitting is a problem for the model. Because the lines stay close together, we can say that overfitting is not a problem. We also do not see a drawn-out plateau in either chart. This indicates that we have properly tuned the model and that accuracy is relatively high. We can see that we have high accuracy, and a low loss value and loss is decreasing, and accuracy is increasing. These are good signs in the evaluation phase.

Difficulties/Limitations:

Some of the difficulties I faced during the creation of this model was the time it took to run each iteration of the model. Because Neural Networks are so complex and expensive in training, high powered GPUs are recommended for training these models. My older computer does not have these, so it took over an hour to run some of the iterations of the model.

Limitations for project come from the lack of data available. If the training set were to be larger than I am sure I could have achieved an even higher accuracy. Also, the goal of this assignment was accuracy and not efficiency, so running time and complexity were not considered. This could result in problems in applying the model if it needed to be applied anywhere. Furthermore, this model is tuned for the clean data that was used for training and testing, if it were to encounter noisier data, the model would likely not perform as well as it has been tuned for clean data.

Future Improvements:

If I were to suggest future improvements, I would probably go along the lines of improving efficiency and reducing complexity to make the model more usable in the real world. With a faster model we can use the model for more applications. The goal of optimization is performance and efficiency and this model lacks in efficiency so the next step would be attempt to get as high of performance with a faster model than the one given above. This can be done by removing layers to reduce complexity, put overfitting measures into the model instead of the training structure (batch size and epochs), etc. Although this is a great model for the project, it is not great for an application anywhere because of the lack of efficiency.