Description of Experiment

For this project we experimented with different optimizers, numbers of epochs, and different types of activation functions for dense layers. **For all the ANN experiments we produced a confusion matrix, a loss plot per class, a recall plot per class, and a model accuracy comparison graph.**

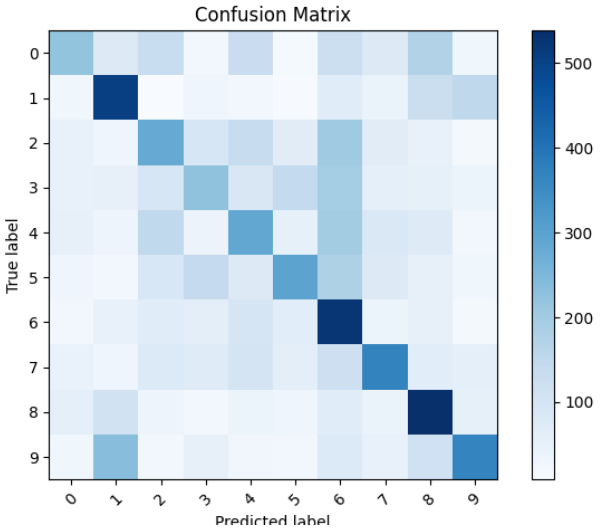
The general architecture of our model consists of the following parameters:

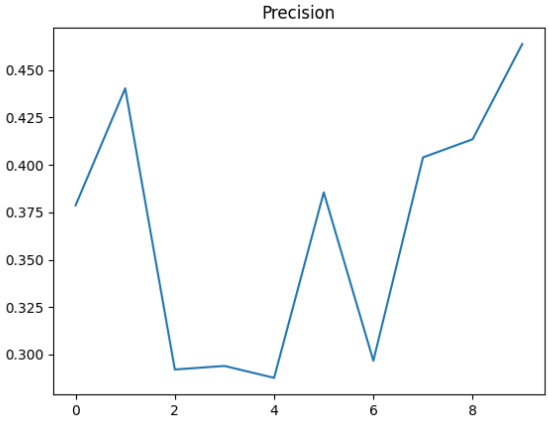
- **A scale factor** of 255 represents all colors that a single pixel can hold.
- **Batch size** of 32 as a way of creating mini-batches since each image is 32-bit.
- Flatten() is called before the output layer to decrease values being passed.

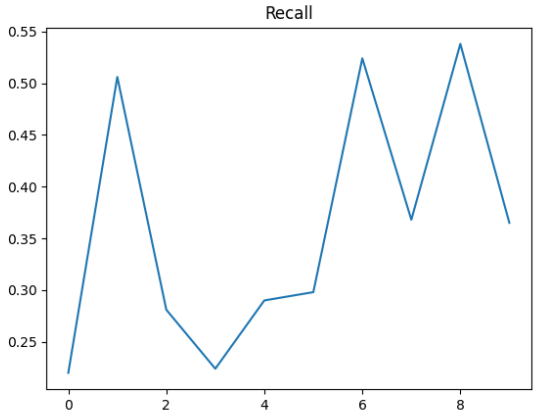
All the parameters above were constant during the experiments. For extensiveness purposes, we will go over one of our first trials and compare that with our best model so far.

First trial

The first experiment consisted of running **6 dense layers of ReLu with N=255, SGD optimizer, 100 epochs.** The results were satisfactory; however, the execution time was too long. The following results were collected:







The diagonal precision is already noticeable, but the accuracy varied a lot per dataset. We can also correlate recalls rate with precision. In other words, precision is higher when recall is higher.

Best model created

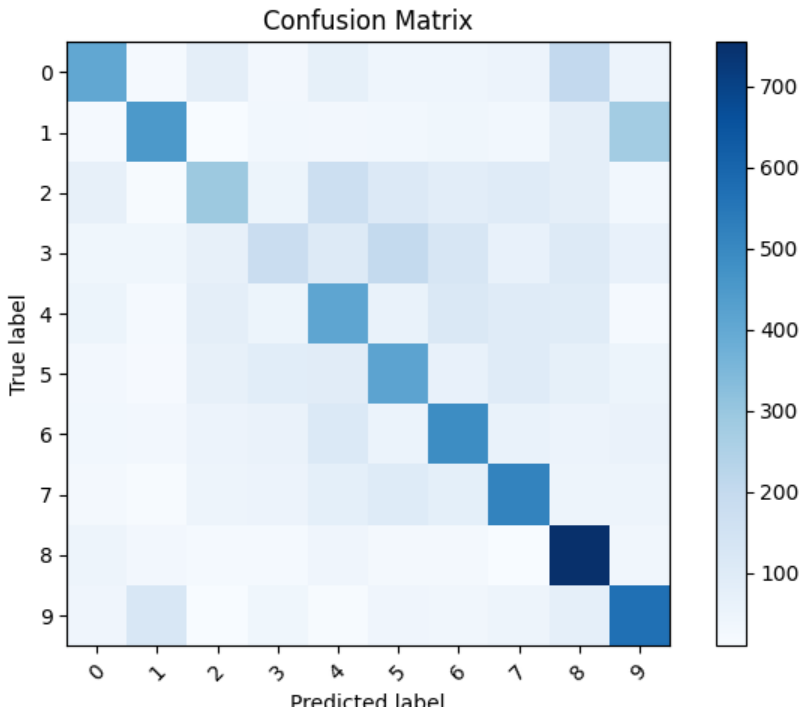
Our better model was firstly based on choosing the best optimizer. According to Keras documentation, **the best optimizer for our purposes might be the RMSprop** (Optimizers (keras.io)). This can potentially increase the learning per epoch exponentially until it reaches a plateau. After that, we also **decreased the number of epochs to 50** to improve execution time. For last, instead of using only ReLu, we also did use of **Tanh and SeLu activation functions**, with decreasing Ns (from 512 until 32, dividing by two in each layer).

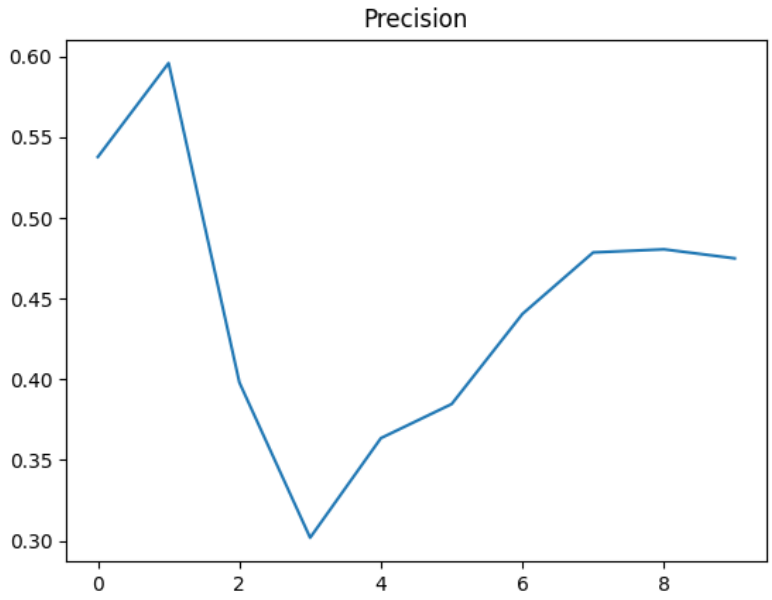
For better visualization, this was the code for the best model layers:

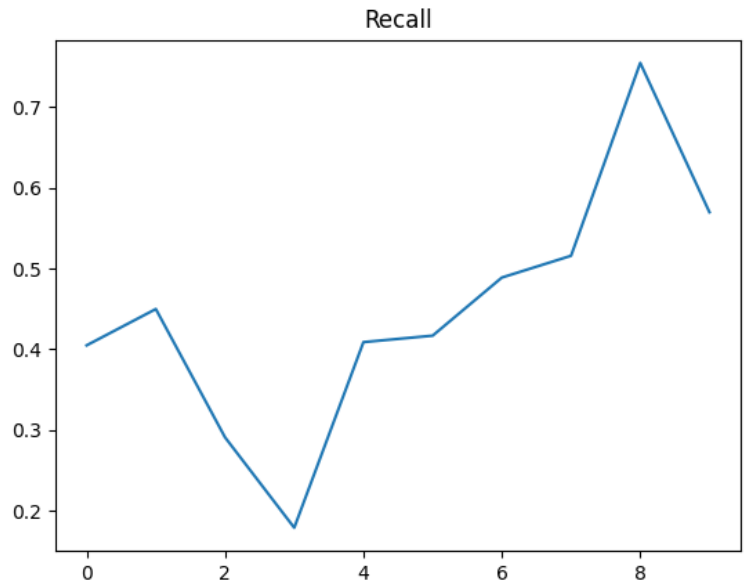
```
model.add(Dense(512, activation='relu', input_shape=(32, 32)))
model.add(Dense(256, activation='selu'))
model.add(Dense(128, activation="tanh"))
model.add(Dense(64, activation="relu"))
model.add(Dense(32, activation='relu'))
model.add(Dense(num_labels, activation='relu'))
```

```
model.add(Flatten())
```

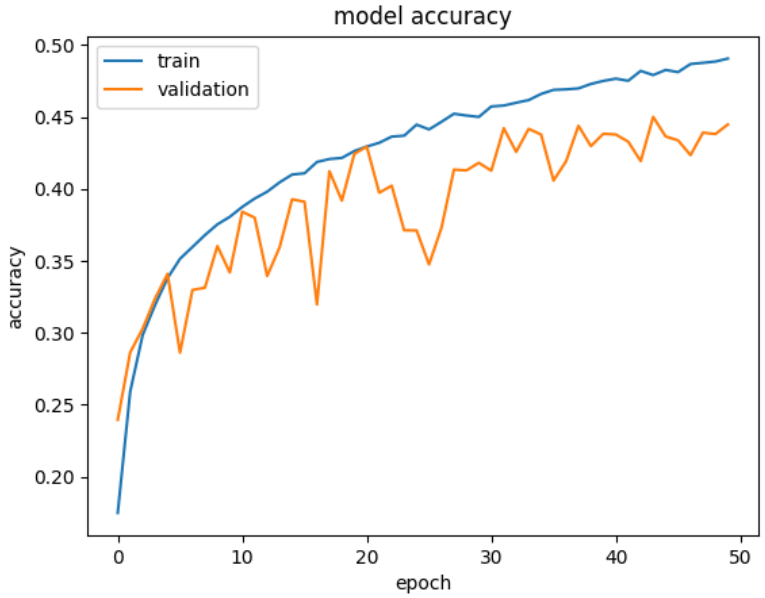
With all that implemented, the accuracy of our model reached 44% (average). The following data was collected:







Recall rate per dataset



Training performance

Average precision per class

0	1	2	3	4	5	6	7	8	9
53.78%	59.6%	39.8%	30.1%	36.35%	38.46%	44.05%	47.86%	47.96%	48.05%

Average recall per class

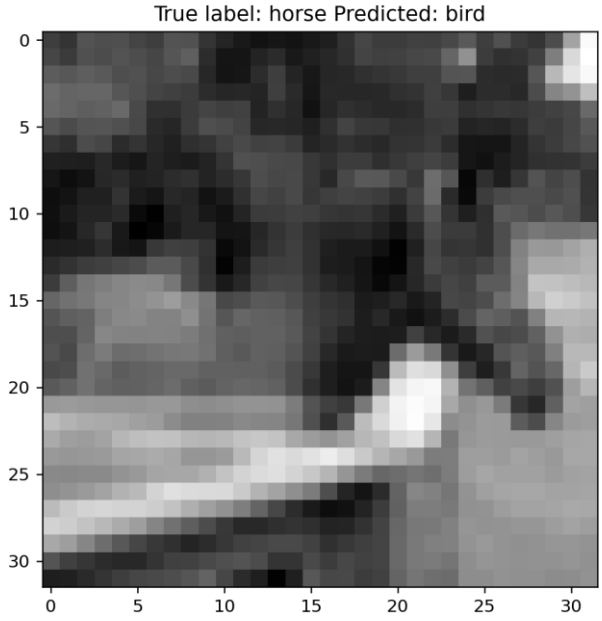
0	1	2	3	4	5	6	7	8	9
40.05%	45%	29.1%	17.9%	40.9%	41.7%	48.9%	51.16%	75.5%	0.57%

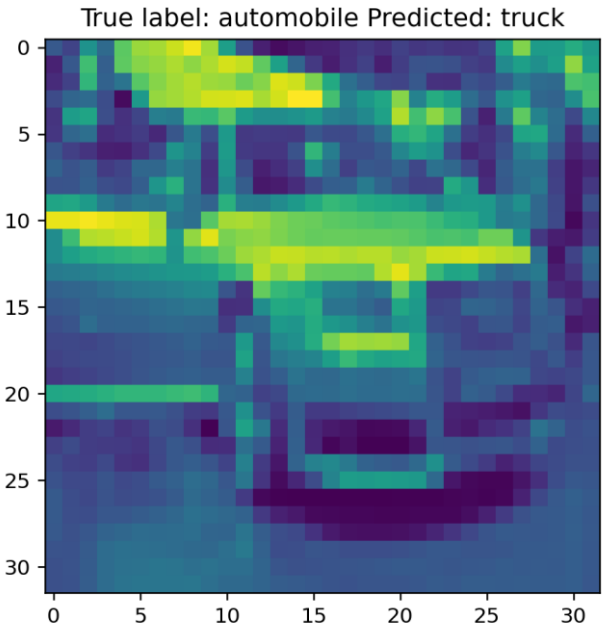
After analyzing the results of the model, we can say that the correlation between precision and recall still holds. We can also infer that some classes are harder to identify than others. Finally, the optimizer and changes regarding the activation functions were also beneficial for the overall ending result.

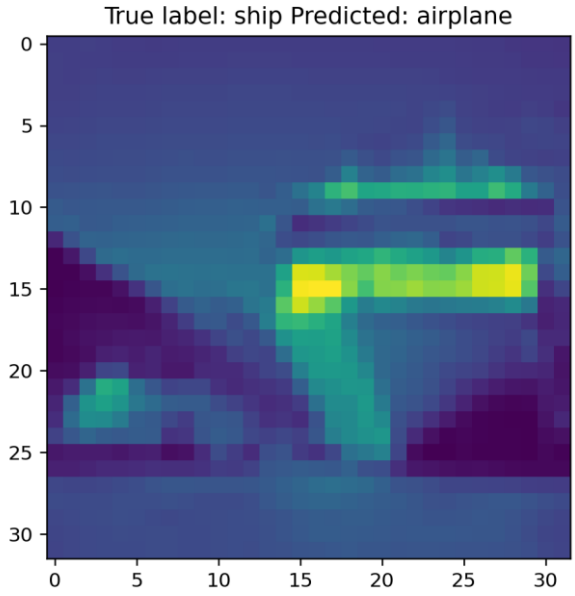
About some of the design choices, after analyzing the pros and cons of each activation function described on this website, we concluded that:

- ReLu is the best performer for large computations, but it may create dying nodes.
- SeLu is a better version of ReLu when it comes to learning.
- Tanh has the advantage of working with negative values (decrease dying nodes)

Visualization of misclassified images:


True label: horse Predicted: bird
0
5
10
15
20
25
30
0
5
10
15
20
25
30


True label: automobile Predicted: truck
0
5
10
15
20
25
30
0
5
10
15
20
25
30


True label: ship Predicted: airplane
0
5
10
15
20
25
30
0
5
10
15
20
25
30