UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Faculty of Engineering, Built Environment and Information Technology
Fakulteit Ingenieurswese, Bou-omgewing en Inligtingtegnologie / Lefapha la Boetšenere, Tikologo ya Kago le Theknolotši ya Tshedimošo

Department of Informatics

INDIVIDUAL ASSIGNMENT

Surname	Wood							
Initials	FS							
Student Number	0	4	8	6	9	4	6	1
Module Code	**INF**				7	9	1	
Assignment number	Assignment 2							
Name of Lecturer	Dr Mike Nkongolo							
Date of Submission	2024/10/10							

Declaration:

I declare that this assignment, submitted by me, is my own work and that I have referenced all the sources that I have used.

Signature of Student	Fabio Wood

Contents

ABSTRACT

The following assignment looks at long short-term memory auto encoders for the reconstruction of images using the CIFAR-10 data set. This data set is composed of 60,000 32 x32 pixel images which span over numerous categories. LSTM auto encoders I used in situations where spatial dependencies as well as sequential dependencies in images allow 4 robust reconstructions of images.

The aim of the assignment is to train an LSTM model to reconstruct clean images from various distorted inputs. This allows for the demonstration for the model to retain image features whilst recovering data. The following assignment investigates data preprocessing techniques which consist of reshaping and normalization followed by the Gaussian-blurred Images that the model trains on. The images that are reconstructed are then assessed and evaluated according to the autoencoders performance in various image clarity enhancement as well as noise reduction.

Applying this approach using image compression noise reduction anomaly detection, the visual comparison between reconstructed images and original images allows for insights into the LSTM Auto encoders for image enhancement in a variety of applications.

INTRODUCTION

Models that can handle complex image processing such as reconstruction segmentation and classification are all developed through deep learning. Along with various deep learning techniques exist auto encoders that can learn representations of data and at the same time can preserve original input features. By allowing this to happen proves valuable for applications where anomaly detection noise reduction and image compression are evident.

Recurrent neural networks and those are especially designed for long short-term memory networks. I'm mainly used in capturing images data where there are spatial dependencies. LSTM Are able to process the number of sequences of data which allows it to be beneficial for compressing noise inputs and reconstructing images.

The CIFAR-10 data set consists of numerous types of images along with various categories of these images. This data set allows particularly difficult challenge for various machine learning models. This is due to the small size of the images as well as all the class diversities. This data set within the aspects of the Simon 2 further evaluate the performance of these auto encoders in looking into images and then reconstructing them.

The main objective of this assignment would be to train an LSTM auto encoder which in turn learns from a Gaussian-blurred image, and then is able to reconstruct clear original versions. By

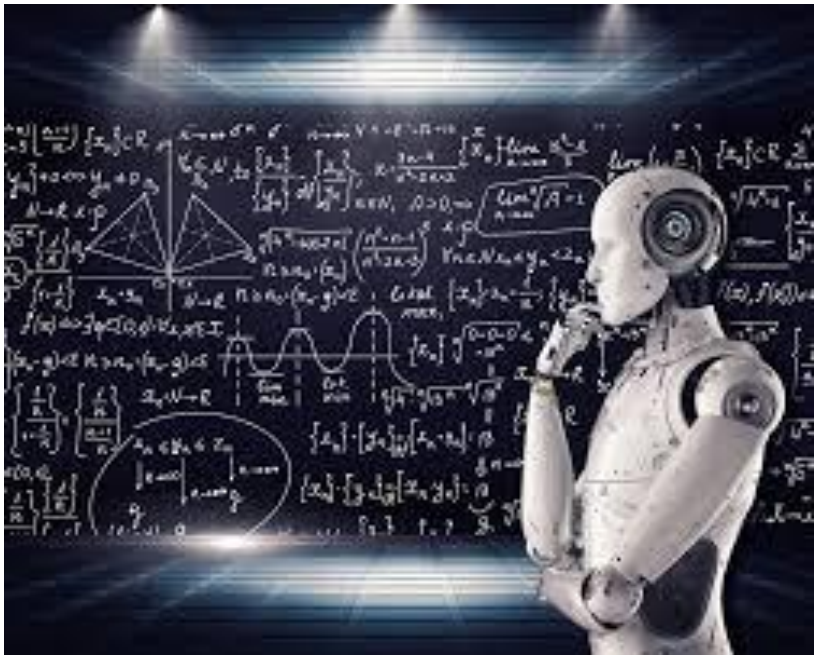
making use of this process the model can demonstrate the ability to analyses image features cover the original input and compress data. By allowing for the reconstructed and distorted images to be compared we can see how the auto encoder offers its effectiveness when there is noise reduction in these images.

LITERATURE REVIEW

Deep learning models such as CNNs and LSTMs have played a vital role in image processing, image classification, reconstruction, and noise reduction. LSTMs networks can capture sequential data and can be applied to image construction particularly when dealing with sequential dependencies and spatial dependencies in pixel data (Wang et al., 2020). However, there are several studies that show that CNNs can perform LSTMs in terms of image classification and image denoising. This is true to their ability to Handle special features (Zhao et al., 2021).

The CIFAR-10 Data set consists of various categories containing very small images and enables challenges to be made four models to be able to classify and reconstruct various images. Krizhevsky and Hinton (2009) that there are numerous difficulties with this data set, and it is crucial to normalize the data as well as reshape it whilst competing various machine learning activities. CNNs can capture spatial hierarchies from images and thus are able to better perform than LSTM networks (Zhao et al., 2021).

Advancements which can combine different architectures such as LSTMs and CNNs in order 2 breakdown complex images and improve overall performance (Nguyen et al., 2022). By offering a hybrid approach, mobile solutions can be generated in handling images with fine-grained pixels or noisy data.



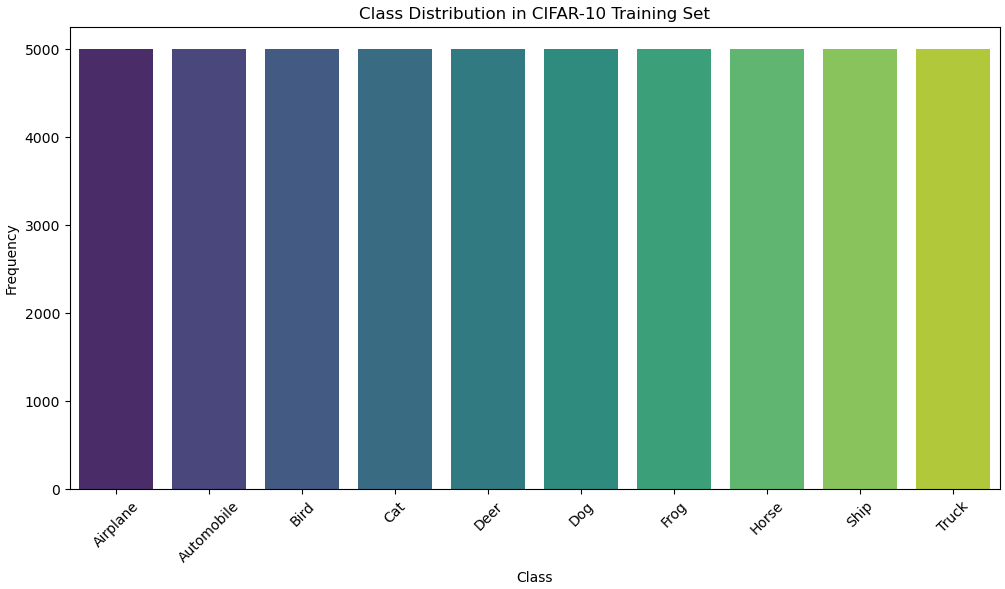
DATA COLLECTION AND PREPARATION

DATA PROCESSING AND EDA

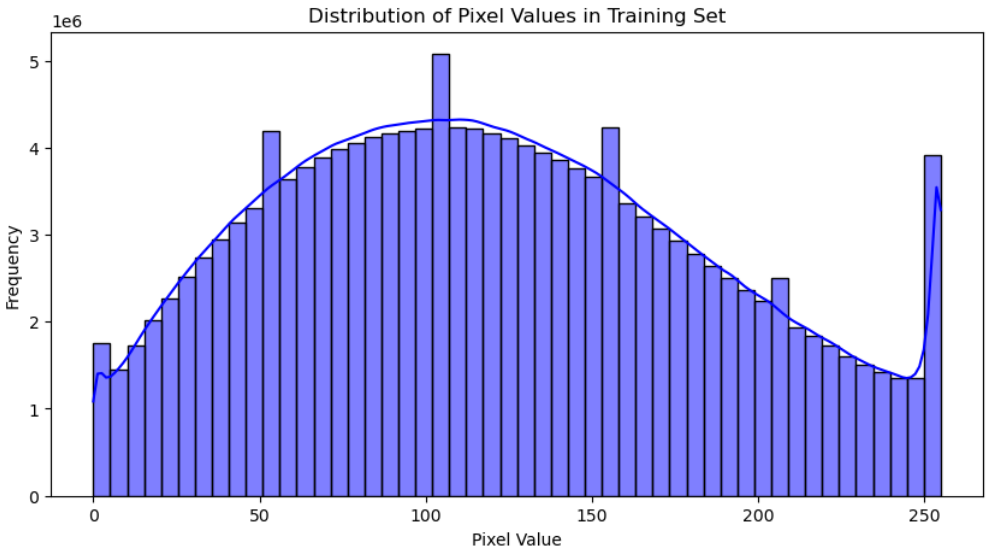
```
Loaded precomputed statistics and mean images from file.
Train images shape: (50000, 32, 32, 3)
Train labels shape: (50000, 1)
Test images shape: (10000, 32, 32, 3)
Test labels shape: (10000, 1)
Basic Statistics of Pixel Values in CIFAR-10 Training Set:
            Statistic  Pixel Value
0                Mean   120.707565
1  Standard Deviation    64.150076
2                 Min     0.000000
3                 Max   255.000000

Per-Channel Statistics of Pixel Values in CIFAR-10 Training Set:
  Channel  Mean Pixel Value  Standard Deviation
0     Red        125.306918           62.993219
1   Green        122.950394           62.088708
2    Blue        113.865383           66.704900
```

The above statistical analysis can provide more information into the CIFAR-10 dataset. This dataset consists of 60 000 32x32 RGB images that are split into 10 000 test images and 50 000 training images. Each of these images are associated with a label that is then linked to one of 10 classes. According to the pixel statistics, the average pixel values are 120.71, and the standard deviation is about 64.15 pixels. This data means that the pixels are spread around this range in the dataset. A minimum and maximum pixel value can be identified. The minimum pixel value is 0 and the maximum pixel value is 255. This is expected however as that is what the values are ranging from between certain limits. The Per-Channel statistics of the pixel values indicate the mean and standard deviations of the pixel values based on the colour channel that they fall int (red, green, blue). By analysing these statistical values, we can see various insights into the dataset which might be of importance when needing to normalize or preprocess any data for the machine learning models.

The bar chart below demonstrates the distribution of all the classes within the CIFAR-10 dataset. Each of the following bars is a representation of the class and the amount of training images for each class. It is evident that each class has a training set of 5000 images. As this is balanced, it enables the machine learning models to be distributed equally. By doing so the models can reduce model biases towards certain classes.

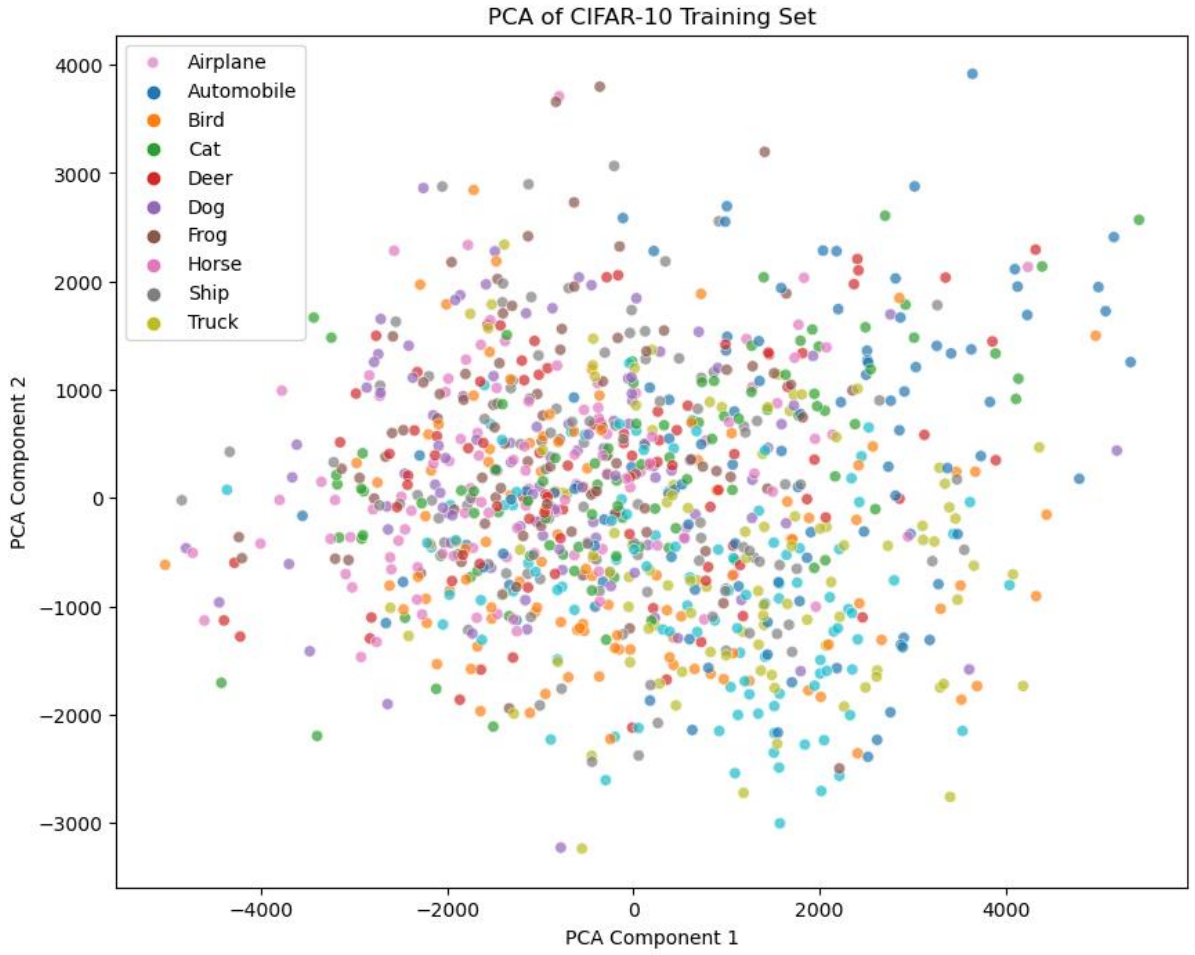


The below histogram can demonstrate how the pixel values are distributed across all the images within the dataset. The curve of the bars can represent the kernel density estimates. According to the graph, it is evident that the pixel values are spread evenly between the range, however there are peaks at the 0, 50, 100, 150, 200 and 255 pixel values. Based on the following distribution, it is evident that the dataset contains a very wide range of brightness levels for the images. This is because there are pixels in both the low and high values.



The scatterplot below indicates a PCA (Principal Component Analysis) for the CIFAR-10 Training Set. This scatterplot can show a two-dimensional representation of the CIFAR-10 dataset once the dimensions are reduced. Each of the points in this diagram represents an image, and the colours are used to differentiate between the different classes that the images are linked to. The points overlapping suggest that the dataset is complex, and linear reduction with PCA won't work for separating classes. The reason for not selecting to use a correlation matrix is because it would

not work with this dataset, When I created a correlation matrix it was derived from the class counts, thus not offering very valuable correlations and insights.



The CIFAR-10 Data set provides access to preprocessed data sets for machine learning applications. This data set consists of 10 categories, with each image being a size of 32x32 pixels, and 60,000 color images in total. The data set is divided into training images and test images. There are 50,000 training images and 10,000 test images. These images are used for the evaluation of the model. Enabling this data set to be loaded into my code allows for testing sets to be available to complete model training and preprocessing.

Load the CIFAR-10 Dataset:

```
# Load the CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

NORMALIZE THE DATA

The images inside of the data set range from zero to 255 – as far as RGB images go this is considered standard. To make the model training faster and more efficient, the pixel values we're normalized to a range of [0,1]. This was done by dividing each pixel value by 255. Buy alarm for

the normalization of the data, enables the learning process to be more stable. This is due to the reasons of smaller values stopping large weight updates whilst the model is being optimized.

Normalize the Dataset:

```
# Normalize the data (CIFAR-10 has 32x32 RGB images)
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
```

RESHAPE THE DATA

For the LTSM model to process correctly and at its optimal performance, it is important that the images are reshaped into a sequence that is suitable for the LTSM autoencoder to handle. This is due to the LTSM autoencoder receiving input data in the form of sequences, thus the 32x32 images needed to be reshaped. This is especially true for images that were in a 3D format, having these images being transformed into a sequence of pixels. Each images' row was indicated as a timestep, thus resulting in a sequence length. This sequence length consisted of 32 timesteps with 96 features per timestep. By doing this, the LTSM network can allow for the processing of the images as a sequence of pixel rows instead of and image made of 2D.

Reshape the Dataset:

```
# Reshape data for CNN (32x32x3) and RNN (32 timesteps, 32*3 features)
x_train_cnn = x_train.reshape(-1, 32, 32, 3)
x_test_cnn = x_test.reshape(-1, 32, 32, 3)
```

METHODOLOGY

To get the LTSM autoencoder to work, the need for the model to have an encoder and decoder was needed. Having a structure such as this one allows for the image to be reconstructed from the compressed input image.

ENCODER

The encoder can compress the input image. This is done by converting the 2D image into a sequence of pixel rows. Inside of this encoder is where the LTSM layer can process sequences. It then captures important features from the sequence and is then able to compact this into a latent representation.

The LTSM layer is fed with the reshaped images. The LTSM layer then learns the sequential relationships based on the pixel data. The spatial dependencies are then processed by each row of the image and recorded as a timestep in each of the sequences. The output of this encoder is

a latent vector that can capture all the necessary information that is in the image. This vector is smaller than the original input. This is where the data is compressed, but still able to retain the most important pieces of the image.

DECODER

This decoder can reconstruct the original image from the latent representation. This is because the decoder can use the LTSM layer to expand the latent representation back to how the image was originally.

This latent representation is then passed on to a second LTSM layer. For the image to be reconstructed, the decoder can repeat the latent vector for each of the timesteps, of which there are 32. This then produces a sequence that can resemble the original pixel structure of the image.

There is a TimeDistributed later that is then applied to the output to make the pixel values for each timestep to line up and correspond with the original rows and features from the original image. The reconstructed sequence is then able to be reshaped back into its original format to get the find output, which is the reconstructed image.

Making use of the LTSM layers in both the encoder and decoder allows for the model to learn spatial dependencies based on the image at hand. It is then able to compress it into a smaller latent representation and then reconstruct the original image with minimal loss of the features that are important.

COMPILE THE MODEL

Before the LTSM autoencoder was trained, the model that I made use of was compiled by using the MSE (Mean Squared Error) loss function. This MSE is suitable for the reconstruction of images tasks as it can calculate the average of the squared differences between the pixel values and the predicted values. This in turn penalizes any large deviations. This makes sure that the model can minimize any reconstruction errors by being able to focus on the reconstructed image and getting it to be as close as possible to the original image.

The Adam optimizer was also used within my code, and this is because it is used to update the weights of the model throughout the duration of the training. This optimizer can combine the advantages of another two optimizers, namely the AdaGrad and RMSProp. By using the Adam optimizer, the learning rate at which the training takes place is both stable and efficient. This in turn assists the model in converging faster to provide a greater solution.

Using the Adam optimizer and the MSE for the loss:

```
# Compile the model using Mean Squared Error loss
cnn_autoencoder.compile(optimizer='adam', loss='mean_squared_error')
```

The Adam optimizer for the CNN model:

```
cnn_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
cnn_history = cnn_model.fit(x_train_cnn, y_train_cnn, validation_data=(x_val_cnn, y_val_cnn), epochs=10, batch_size=128)
```

The Adam optimizer for the RNN model:

```
rnn_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
rnn_history = rnn_model.fit(x_train_rnn, y_train_rnn, validation_data=(x_val_rnn, y_val_rnn), epochs=10, batch_size=128)
```

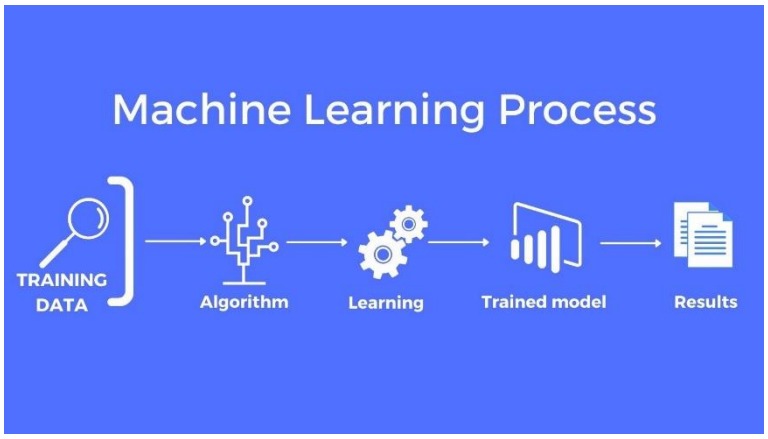
TRAIN THE MODEL

The model has been trained by having inputs (being the distorted images) and the target (the clear images) as the training data.

The input, being the distorted images, had the Gaussian blur applied to the original CIFAR-10 dataset images. Although these images were blurry, they were able to serve as the inputs and mimic majority of real-world scenarios whereby most images are either noisy or not of good quality.

The target outputs were the clear images from the dataset. The goal was for the LTSM autoencoder to reconstruct clear images through the training of distorted inputs. This was done by reducing the MSE loss.

Throughout the process of training, the model adjusted the weight to reduce the difference between the original clear images and the reconstructed images. By having these distorted images for the input images and the clear images as the output, the autoencoder was able to denoise the images, thus ultimately improving the quality. I had some difficulties in terms of my machine for the processing of the images, thus I only used 10 epochs with a batch size of 128. Although if I had increased the epochs and the batch size the results would most likely have been better.



```
CNN Autoencoder Summary:

Model: "sequential_7"
```

Layer (type)	Output Shape	Param #
conv2d_11 (Conv2D)	(None, 32, 32, 32)	896
max_pooling2d_5 (MaxPooling2D)	(None, 16, 16, 32)	0
conv2d_12 (Conv2D)	(None, 16, 16, 32)	9,248
up_sampling2d_4 (UpSampling2D)	(None, 32, 32, 32)	0
conv2d_13 (Conv2D)	(None, 32, 32, 3)	867

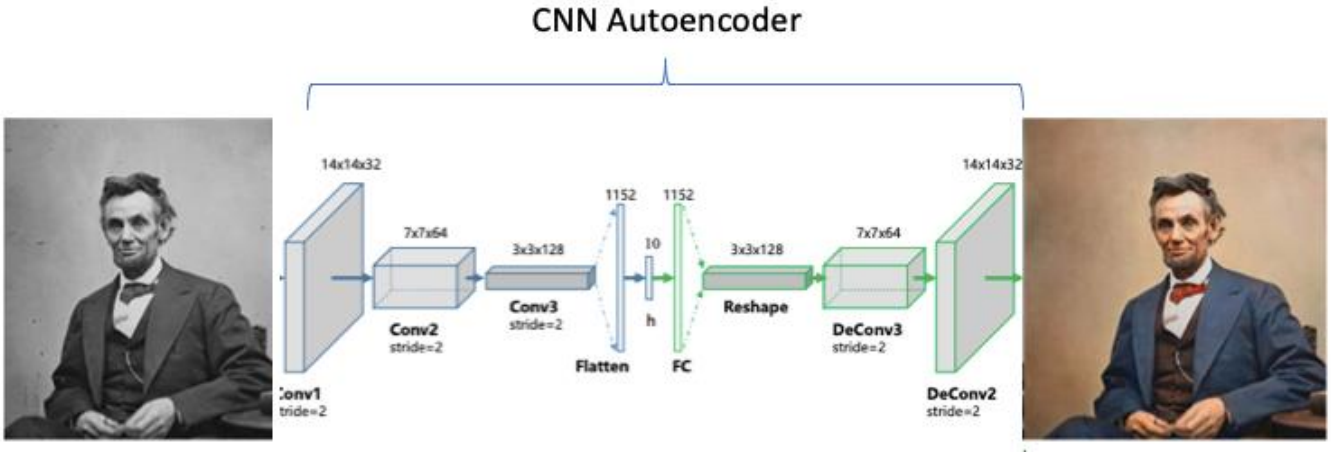
```
Total params: 11,011 (43.01 KB)

Trainable params: 11,011 (43.01 KB)

Non-trainable params: 0 (0.00 B)
```

The above output demonstrates a summary of the CNN Autoencoder model which consists of 5 layers. The Conv2D layer consists of 32 filters and a kernel size of 3x3.

The MaxPooling2D later is responsible for reducing the spatial dimensions by half. This means that it was originally 32x32 and now it is 16x16. The Conv2D layer consists of 32 filters. The UpSampling2D layer is responsible for restoring the spatial dimensions back into a 32x32.

And lastly the Conv2D layer can output an image with the shape that is the same as the input. With the model being able to have a total of 11 011 trainable parameters, it is noted that this consists of the total amount of parameters, this there being zero non-trainable parameters.

The architecture for this model is used for the compressing and reconstruction of images within an autoencoder.



```
RNN Autoencoder Summary:

Model: "sequential_8"
```

Layer (type)	Output Shape	Param #
simple_rnn_5 (SimpleRNN)	(None, 128)	28,800
repeat_vector_4 (RepeatVector)	(None, 32, 128)	0
simple_rnn_6 (SimpleRNN)	(None, 32, 128)	32,896
time_distributed_3 (TimeDistributed)	(None, 32, 96)	12,384

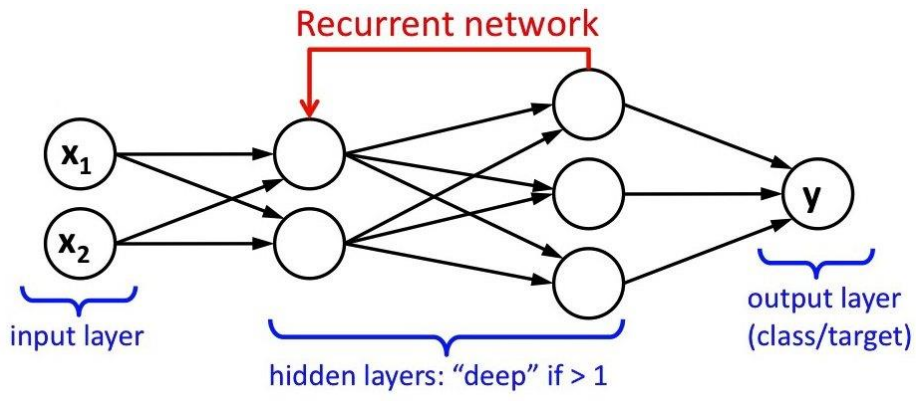
```
Total params: 74,080 (289.38 KB)

Trainable params: 74,080 (289.38 KB)

Non-trainable params: 0 (0.00 B)
```

The above output displays the RNN autoencoder model that is built using various layers of sequence. The SimpleRNN layer is responsible for the output of a latent representation of a size of 128 based on the input sequence.

Moving onto the next sequence, namely the RepeatVector Layer, where this layer can replicate the latent vector 32 times to match the length of the sequence. Another SimpleRNN layer is then able to process the sequence that has been repeated, thus outputting a sequence of 32 timesteps containing 128 features.

A TimeDistributed layer then can allow for a fully connected layer at each timestep, thus producing an output sequence of 32 timesteps containing 96 features per timestep completed. This model demonstrates that there are 74 080 trainable parameters and all of them are used for the model throughout the training process, thus leaving zero non-trainable parameters.

The architecture for this model is to compress and reconstruct sequential data, thus enabling it to be used time-series analysis or sequence modelling.



LSTM Autoencoder Summary:

Model: "sequential_9"

Layer (type)	Output Shape	Param #
lstm_4 (LSTM)	(None, 128)	115,200
repeat_vector_5 (RepeatVector)	(None, 32, 128)	0
lstm_5 (LSTM)	(None, 32, 128)	131,584
time_distributed_4 (TimeDistributed)	(None, 32, 96)	12,384

Total params: 259,168 (1012.38 KB)

Trainable params: 259,168 (1012.38 KB)

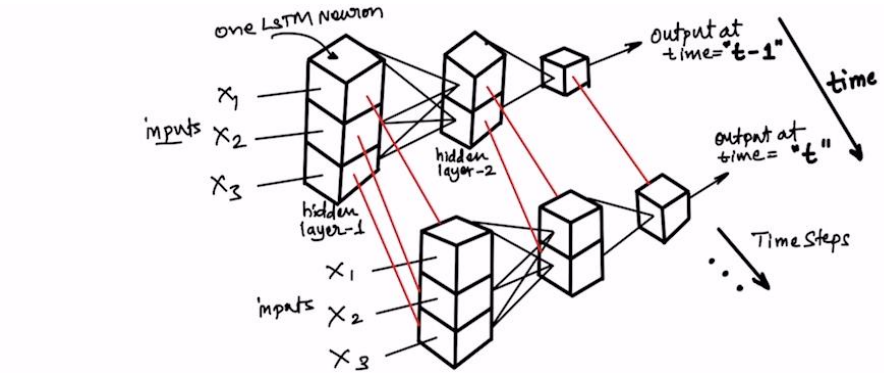
Non-trainable params: 0 (0.00 B)

The above output is for the LSTM Autoencoder model. This model begins with the LSTM layer whereby it outputs a latent representation of 128 in size, thus compressing the data that was inputted.

Following this is the RepeatVector layer that can replicate the latent vector a total of 32 times to match the sequence length of the original. Another LSTM layer follows this that can then reconstruct the sequence, thus outputting a sequence of 32 timesteps.

These timesteps each contain 128 features. Lastly the TimeDistributed layer can apply the fully connected layer to each of the created timesteps. This then can generate a sequence output of 32 timesteps which contain 96 features per timestep.

The model has a total of 259 168 trainable parameters, and that is the total amount of parameters. Therefore, it is evident that there are zero non-trainable parameters. The architecture of this design is to be able to handle sequential data, reconstructor an input sequence through the LTSM layer and to capture temporary dependencies.

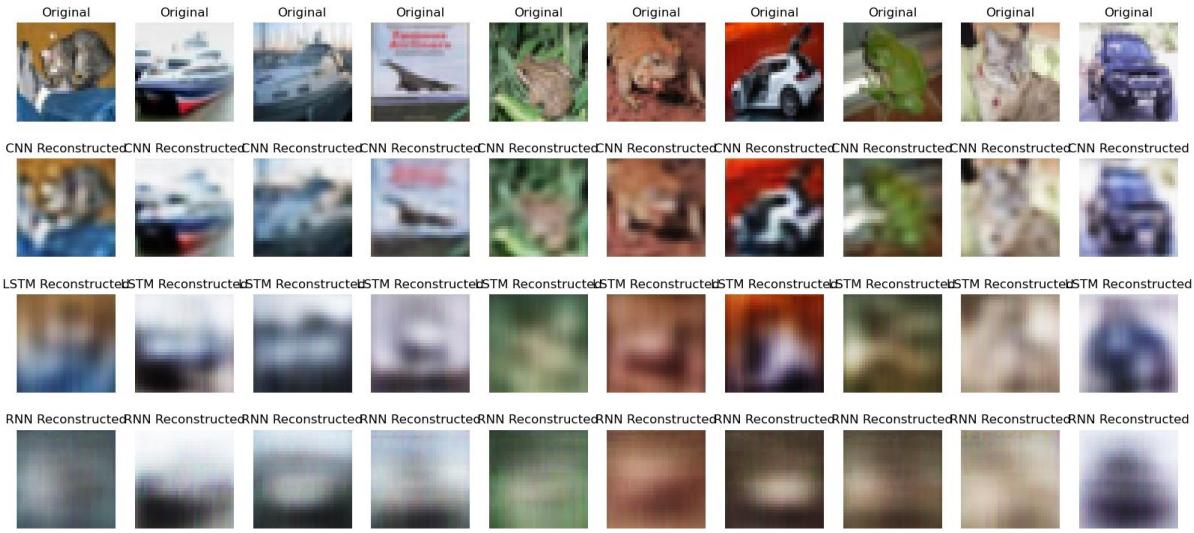


RESULTS

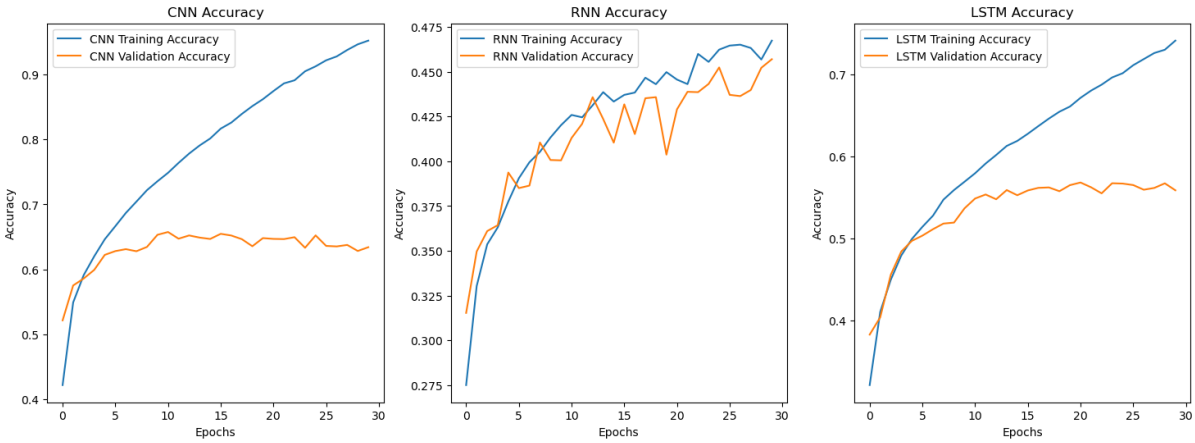
EVALUATE THE MODEL

RECONSTRUCT AND VISUALIZE

In the below image we can see a comparison of the images that are available in the CIFAR-10 dataset and their reconstructions via 3 different models. These models include the CNN (Convolutional Neural Network), RNN (Recurrent Neural Network) and the LSTM (Long Short-Term Memory). The first row in the image contains the original, and then the rows to follow display the model results. Based on this image it is evident that the CNN Reconstruction retains the most details, however they are still a bit blurry. The RNN reconstructions have lost most of the fine details, thus resulting in the images being very blurry. The LSTM reconstructions are more blurred than the CNN, as well as having lost more details than the CNN, however, the LSTMs are usually suited for sequential data rather than images. These results display the importance of CNN in handling spatial data and highlights the limitations of LSTMs and RNNs for the reconstruction of images.



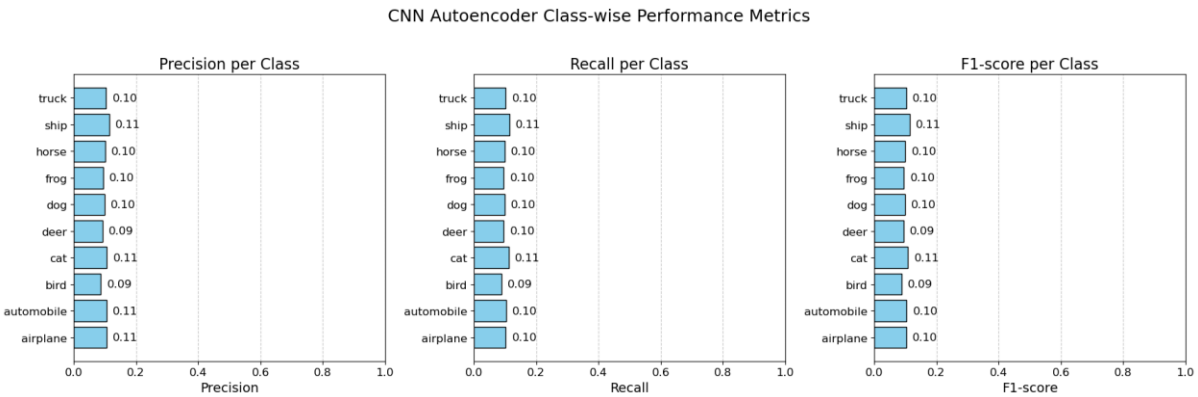
Below are graphs that display the CNN and RNN Accuracies respectively. The CNNs accuracy is nearly 100% after 30 epochs, however, it struggles from overfitting. This is evident from the validation accuracy plateauing around 60%-65%. The RNN model displays an improvement that is much slower than the CNN model. The peak for the RNN model is around the 45% mark. This indicates that the RNNs are much less suited for image data because of the inability to handle spatial dependencies. The LSTM model performs better than the RNN model, however it too shows signs of overfitting. It reaches a training accuracy of around 70% but plateaus at around 55%-60% in the validation accuracy. These graphs indicate that RNNs and LSTMs struggle with image classification in comparison to that of the CNN model.



The CNN Autoencoder Classification Report below indicates that the model has a low performance across all the classes as the precision, recall and F1-Scores range from 0.09 – 0.11, thus indicating it as a low accuracy. The overall accuracy is only 10%. This indicates that the CNN autoencoder struggles with this classification.

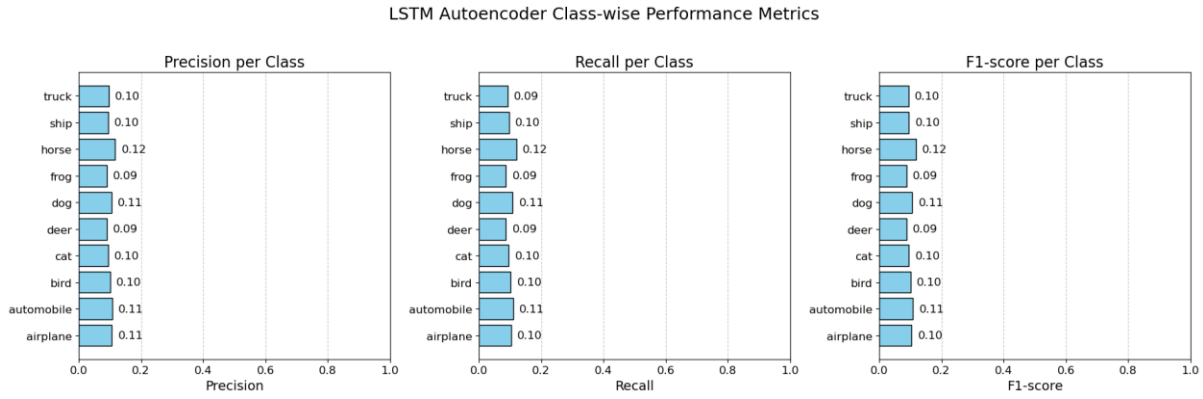
CNN Autoencoder Classification Report:

	precision	recall	f1-score	support
airplane	0.11	0.10	0.10	1000
automobile	0.11	0.10	0.10	1000
bird	0.09	0.09	0.09	1000
cat	0.11	0.11	0.11	1000
deer	0.09	0.10	0.09	1000
dog	0.10	0.10	0.10	1000
frog	0.10	0.10	0.10	1000
horse	0.10	0.10	0.10	1000
ship	0.11	0.11	0.11	1000
truck	0.10	0.10	0.10	1000
accuracy			0.10	10000
macro avg	0.10	0.10	0.10	10000
weighted avg	0.10	0.10	0.10	10000



LSTM Autoencoder Classification Report:

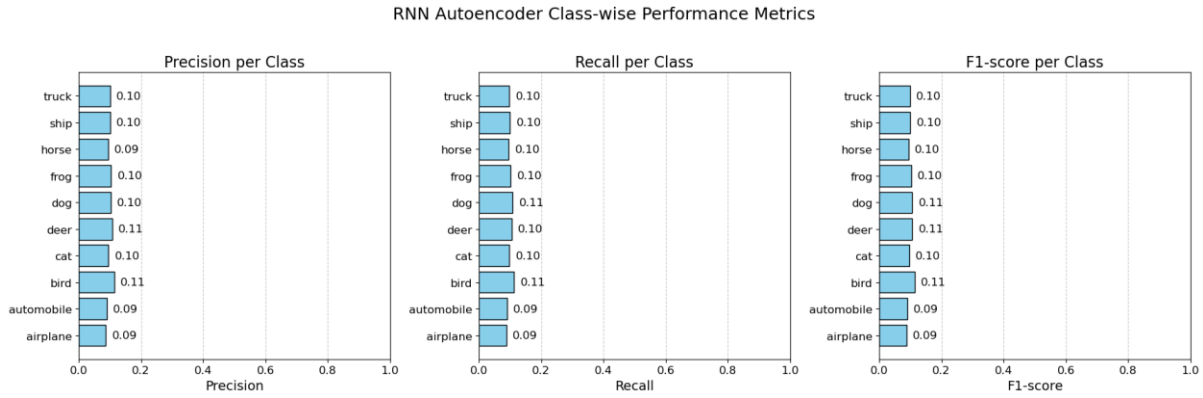
	precision	recall	f1-score	support
airplane	0.11	0.10	0.10	1000
automobile	0.11	0.11	0.11	1000
bird	0.10	0.10	0.10	1000
cat	0.10	0.10	0.10	1000
deer	0.09	0.09	0.09	1000
dog	0.11	0.11	0.11	1000
frog	0.09	0.09	0.09	1000
horse	0.12	0.12	0.12	1000
ship	0.10	0.10	0.10	1000
truck	0.10	0.09	0.10	1000
accuracy			0.10	10000
macro avg	0.10	0.10	0.10	10000
weighted avg	0.10	0.10	0.10	10000



Based on the above results for the LSTM Autoencoder Classification report and diagram, it is evident that the there is a low performance throughout every single class within the dataset. The recall, precision and F1-score were all between 0.09 and 0.12. This indicates that the model is unable to classify images. This is emphasised through a 10% accuracy rate. This data indicates that the model struggles with classification tasks.

RNN Autoencoder Classification Report:

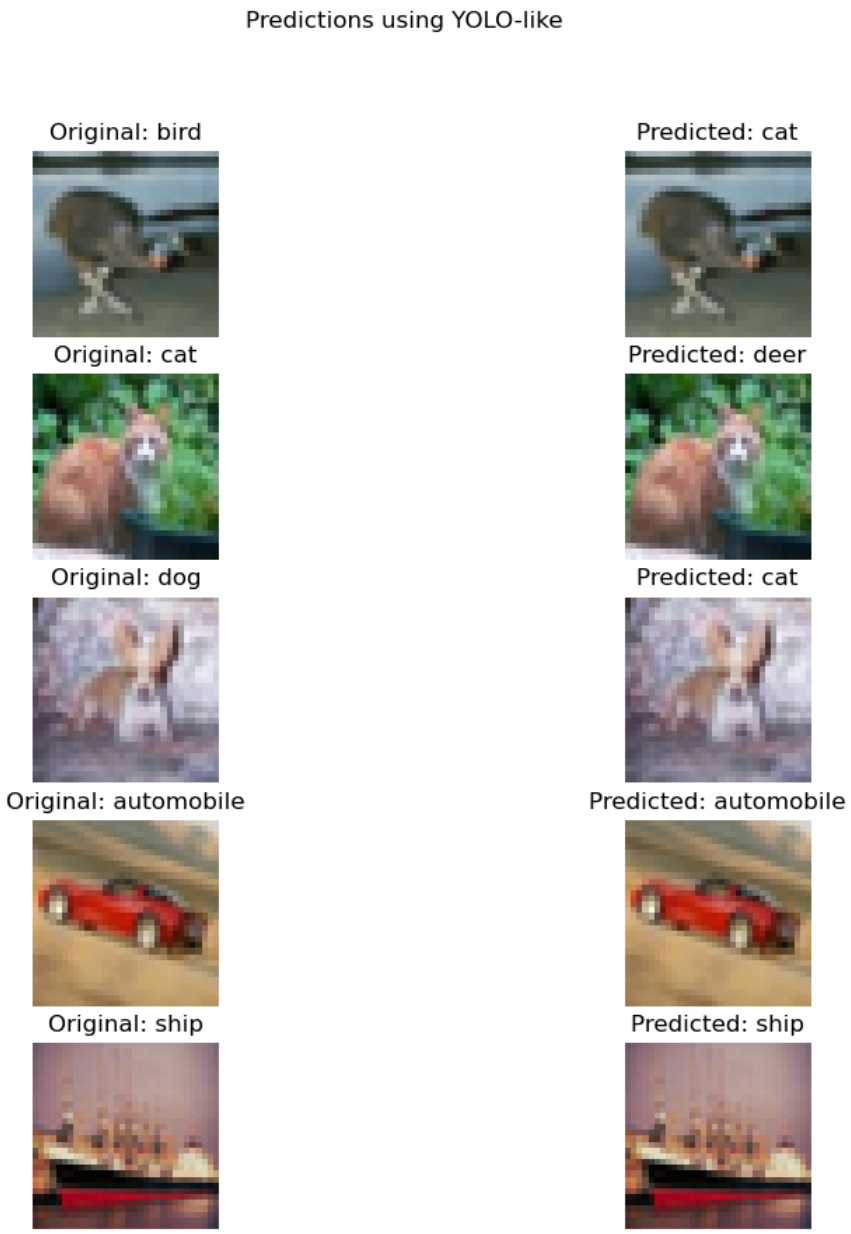
	precision	recall	f1-score	support
airplane	0.09	0.09	0.09	1000
automobile	0.09	0.09	0.09	1000
bird	0.11	0.11	0.11	1000
cat	0.10	0.10	0.10	1000
deer	0.11	0.10	0.11	1000
dog	0.10	0.11	0.11	1000
frog	0.10	0.10	0.10	1000
horse	0.09	0.10	0.10	1000
ship	0.10	0.10	0.10	1000
truck	0.10	0.10	0.10	1000
accuracy			0.10	10000
macro avg	0.10	0.10	0.10	10000
weighted avg	0.10	0.10	0.10	10000



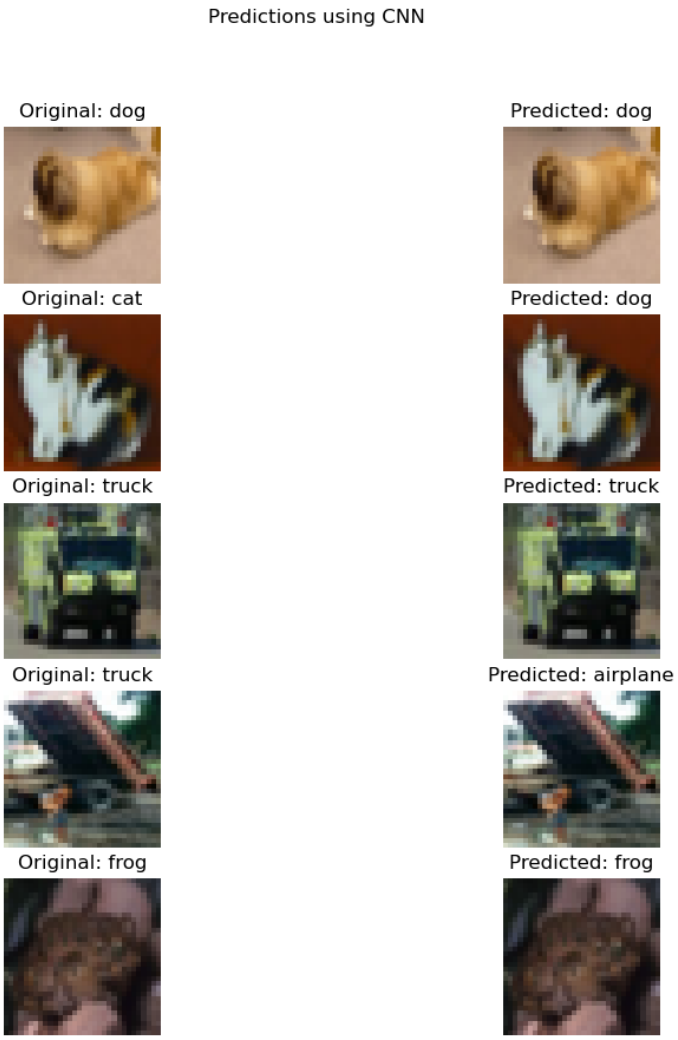
The above graphs analyse the RNN Autoencoder Classification Report as well as the performance metrics. It is evident that it has poor results in performance just like the other models throughout all the classes. The recall, precision and F1-score range between 0.09 and 0.11. This means that the model is struggling at classifying images. With 10% being the overall accuracy, it suggests that it is not an effective model for the classification of images. RNN models are more suited for sequential data.

ANALYTICS USE CASES

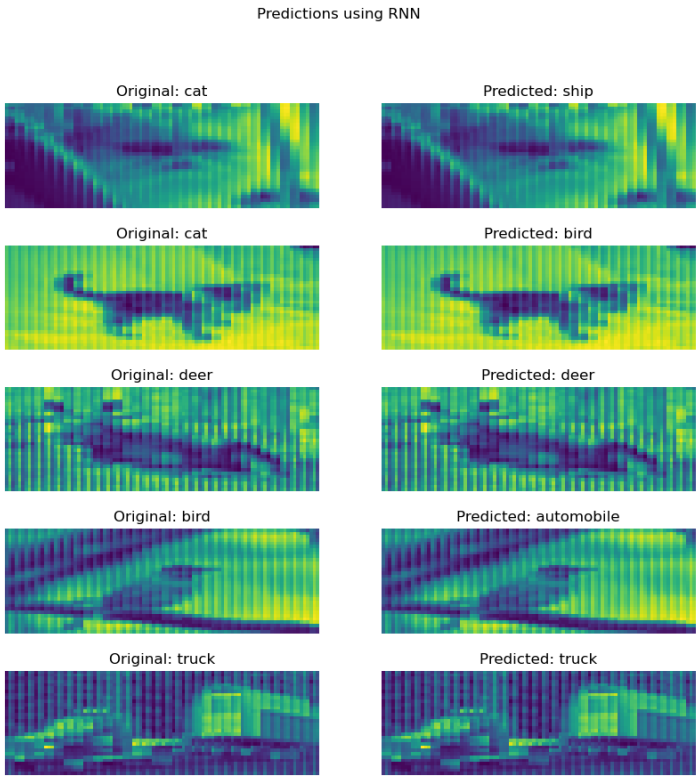
I found a library called YOLO (You-Only-Look-Once). The code can randomly select five images that exist in the CIFAR-10 dataset. It then can display them side by side with the true class and the predicted class. The dataset consists of 10 classes (categories). The left side can display the original image with the correct label, whilst the right side is able to display the models predicted label.



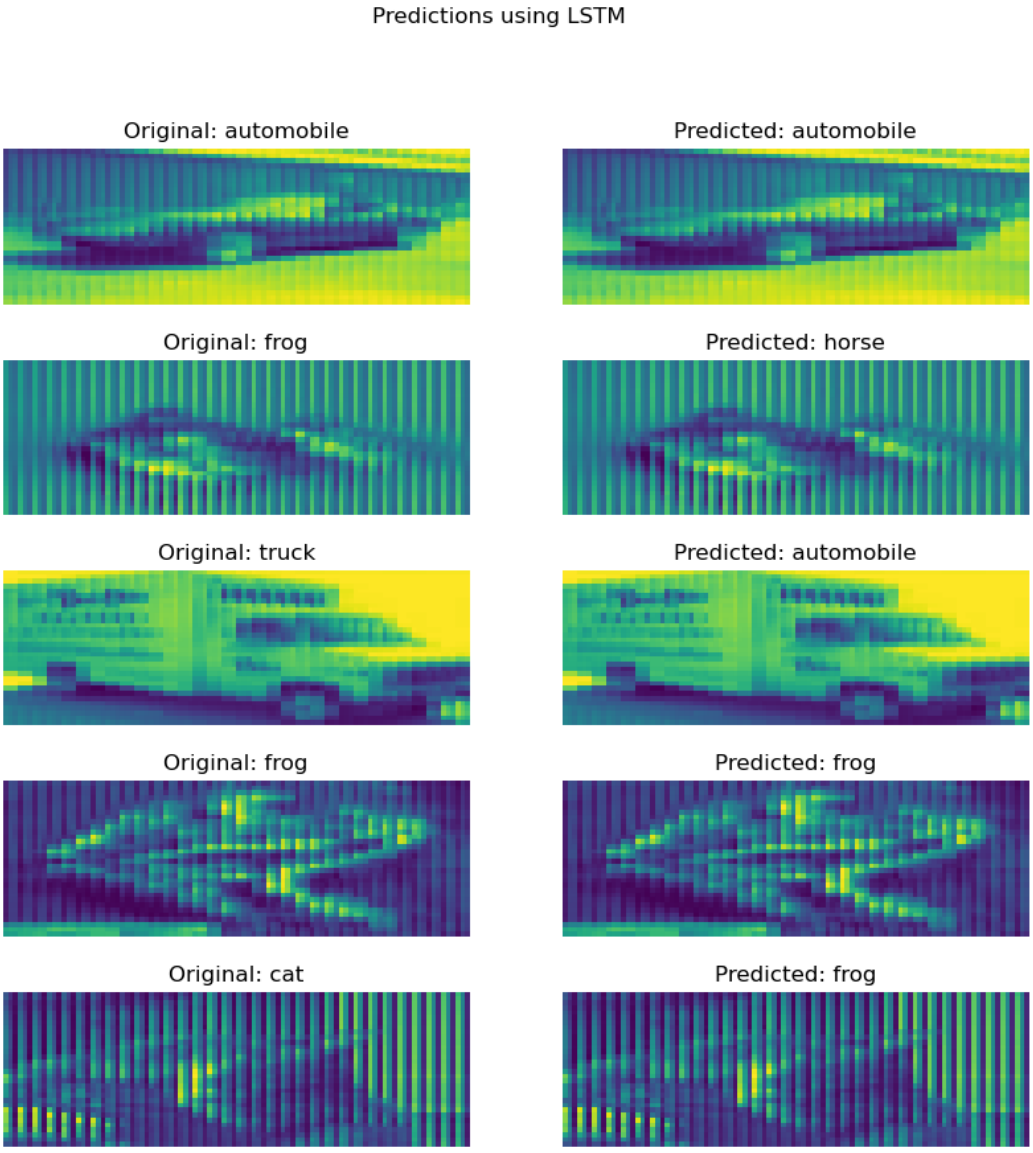
These are the predictions for the YOLO-like model. It is evident that the model has made some errors in the classification. Based on the predictions it seems as though it has better results for vehicles than it does for animals. This could be due to the complexities of the images in certain classes.



These are the predictions for the CNN model. It managed to predict it right on the dog and the truck, however it still struggled with some classifications.
This could be due to noisy or very vague images in the dataset.

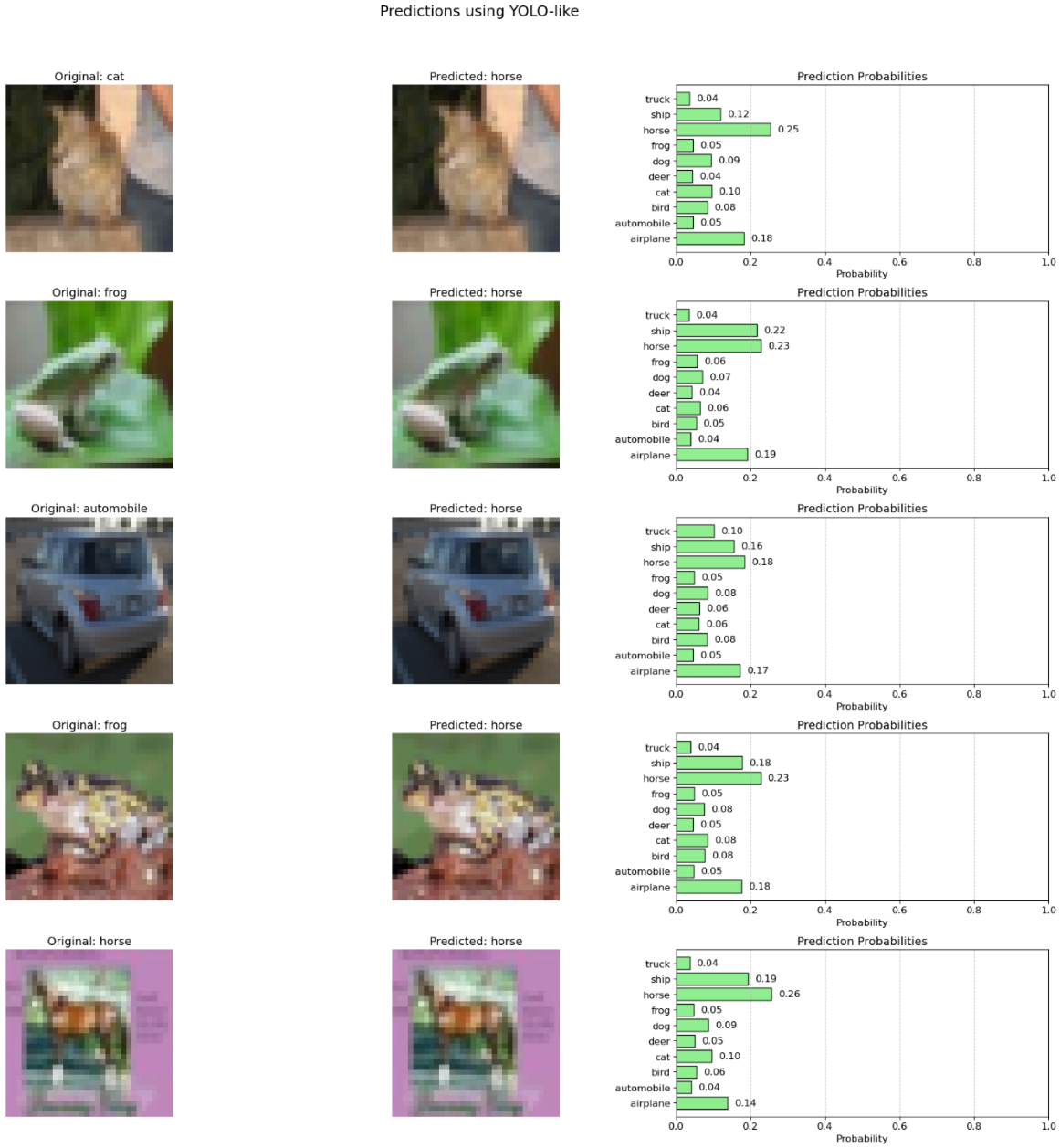


These are the predictions for the RNN model. It managed to predict the deer and the truck, however, it still struggled with some classifications.
This could be because the RNN model is better suited for sequential data rather than data in the form of images.

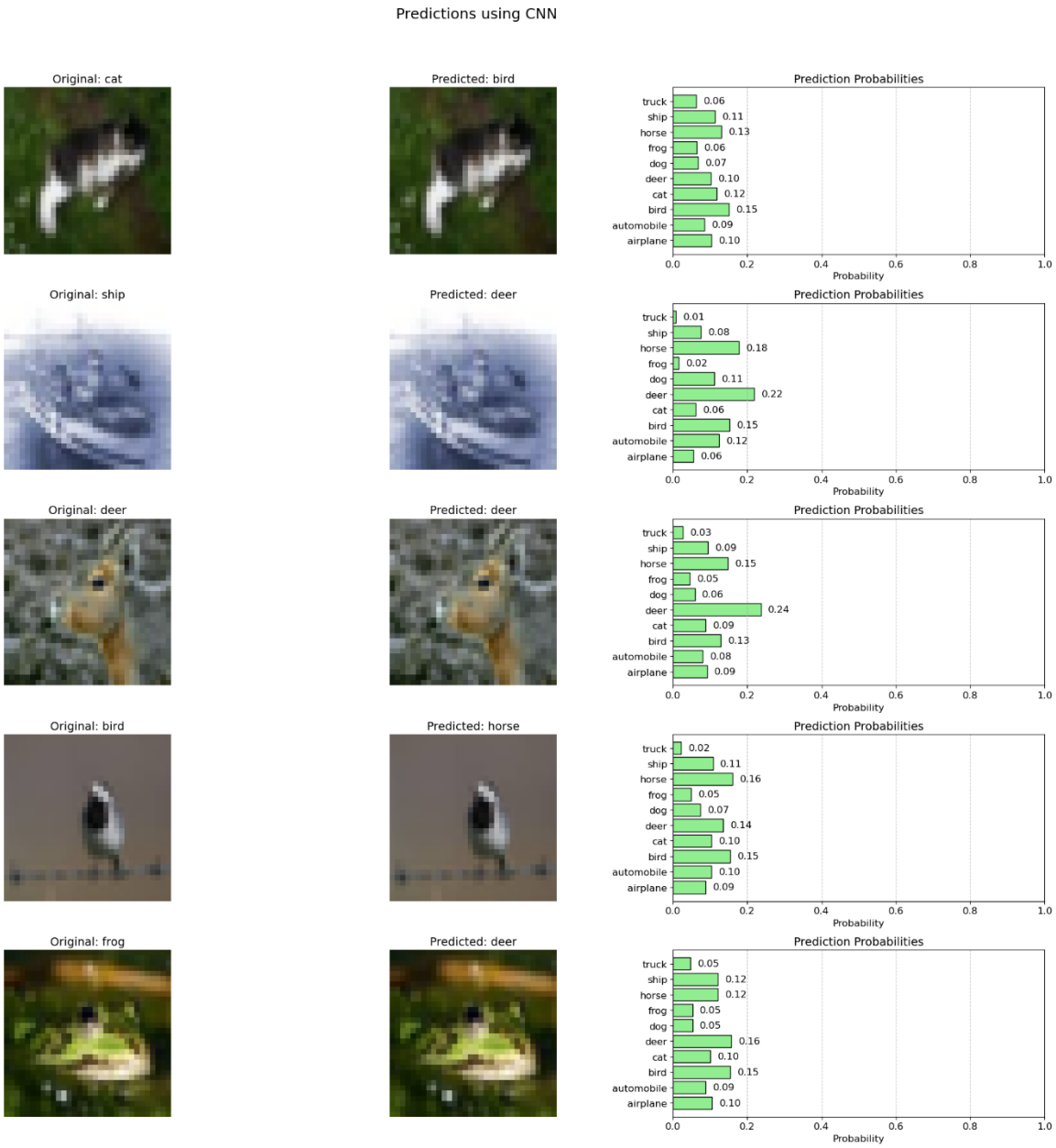


These are the predictions for the LSTM model. It was able to predict the frog correctly however it still struggled with numerous classifications.

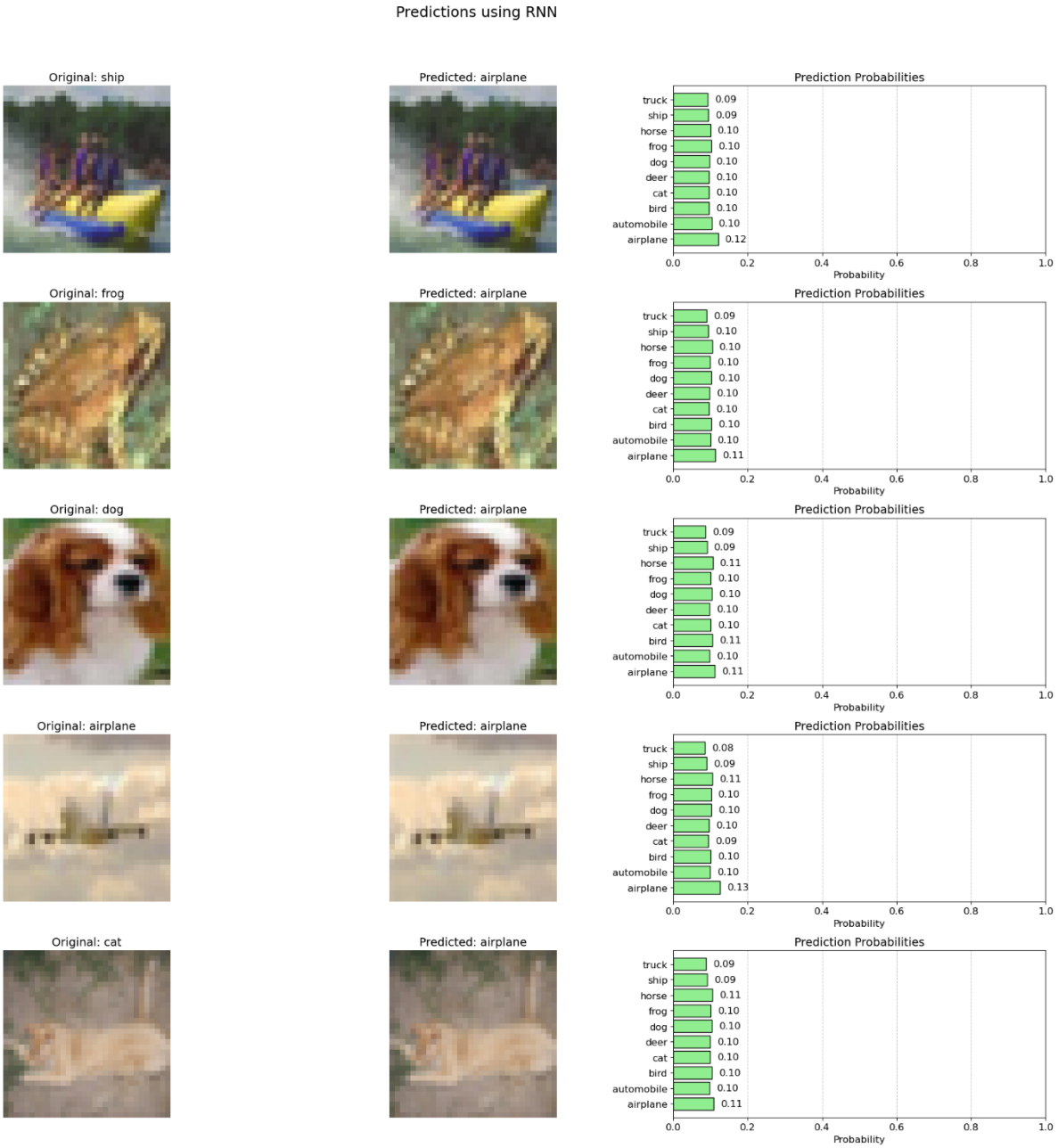
Furthermore, I added the probabilities of the predictions for each class:



The top images indicate the probabilities of the predictions based on the YOLO-Like model for each class based on the image. It marked all the predictions as horse as indicated by the probabilities counts. This in turn suggests that the model has learned a biased pattern or has not been trained correctly. This could lead to poor performance and outcomes. This is a strong indication of overfitting or has issues with certain classes.



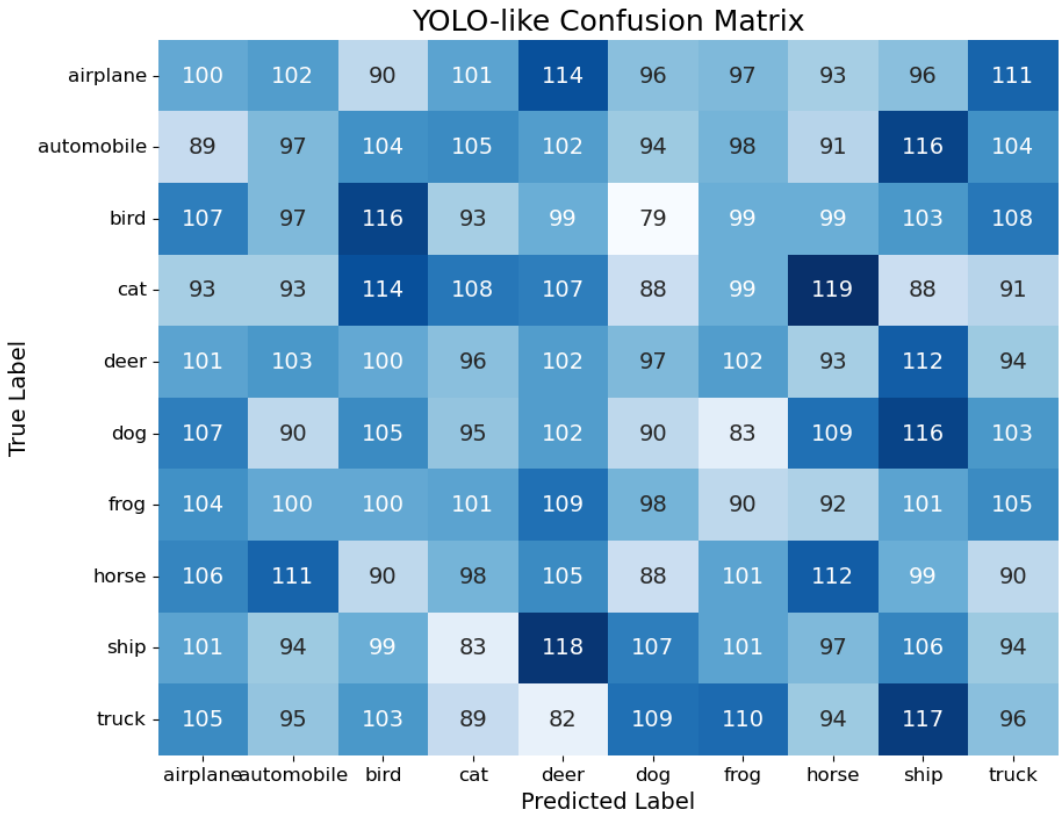
The top images indicate the probabilities of the predictions based on the CNN model. This model can correctly identify some images, however there are several image classifications that are incorrect. This indicates that the model has some uncertainty in a few predictions. Based on the results it could suggest that it has some difficulty in identifying images that have similar looking classes, particularly animals.



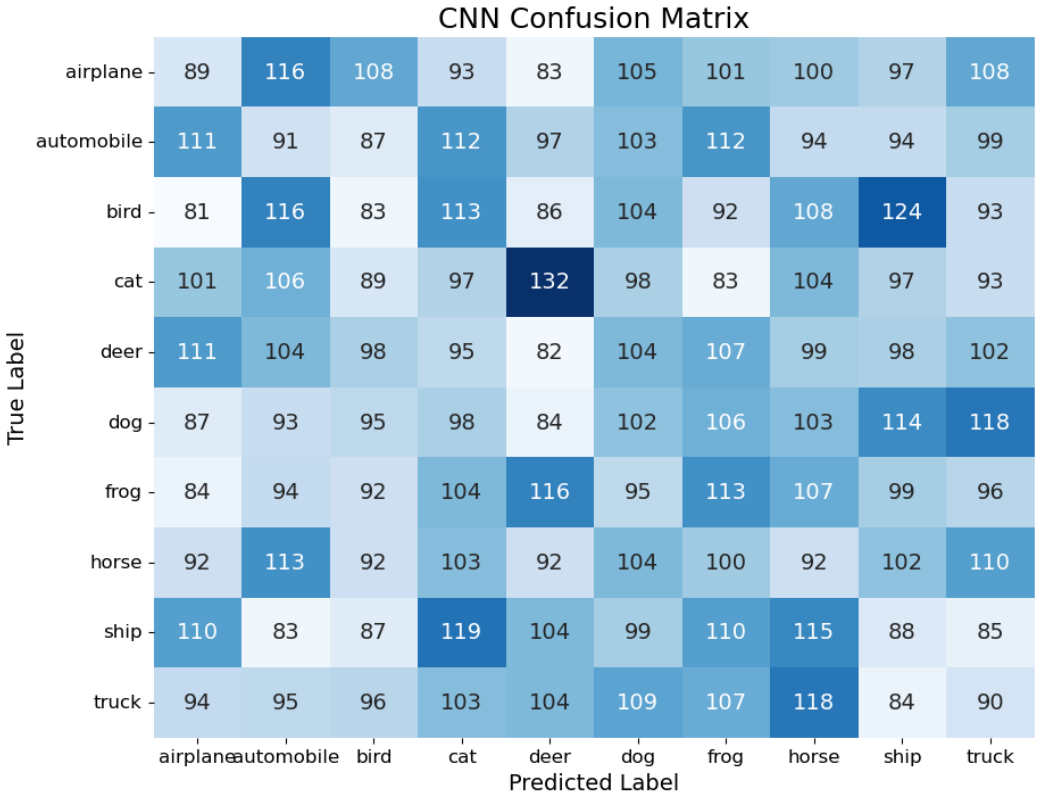
The top images indicate the probabilities of the predictions based on the RNN model. This model predicts airplane as all its predictions. This would suggest that the model has a big issue. The prediction probabilities are very close to uniformly distributed, with a preference to airplanes. This emphasises that the model has poor predictive performance. This could be because this model can't handle image classification, and rather sequential data that than structured data in the form of images.

Predictions using LSTM

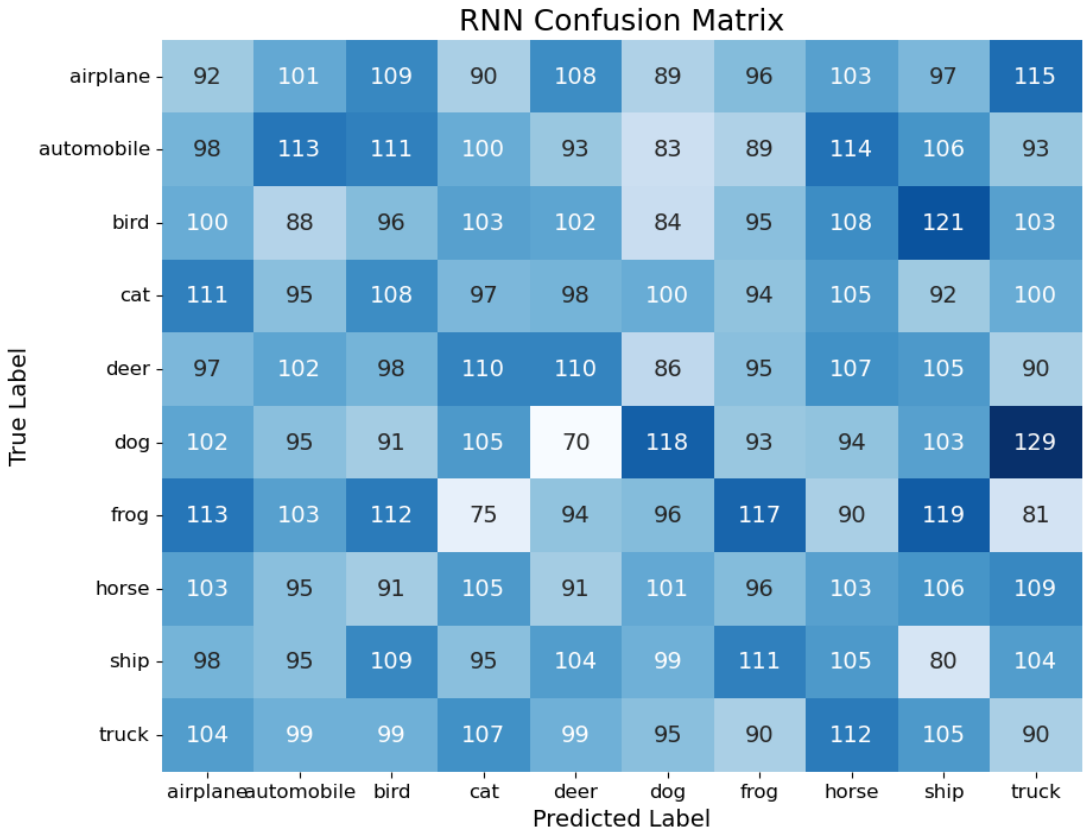
The above images make use of the LSTM model predictions for the dataset. The model seemed to get a bit confused between the deer and the horse. Based on the predictions it seems as though the model is uncertain in its predictions. This is seen through the probabilities that are occurring around the 10% mark for almost every prediction probability. This model has some trouble distinguishing between objects and animals. The output emphasizes that the model is able to handle sequential data but not perform as well as the CNN model for image classification.



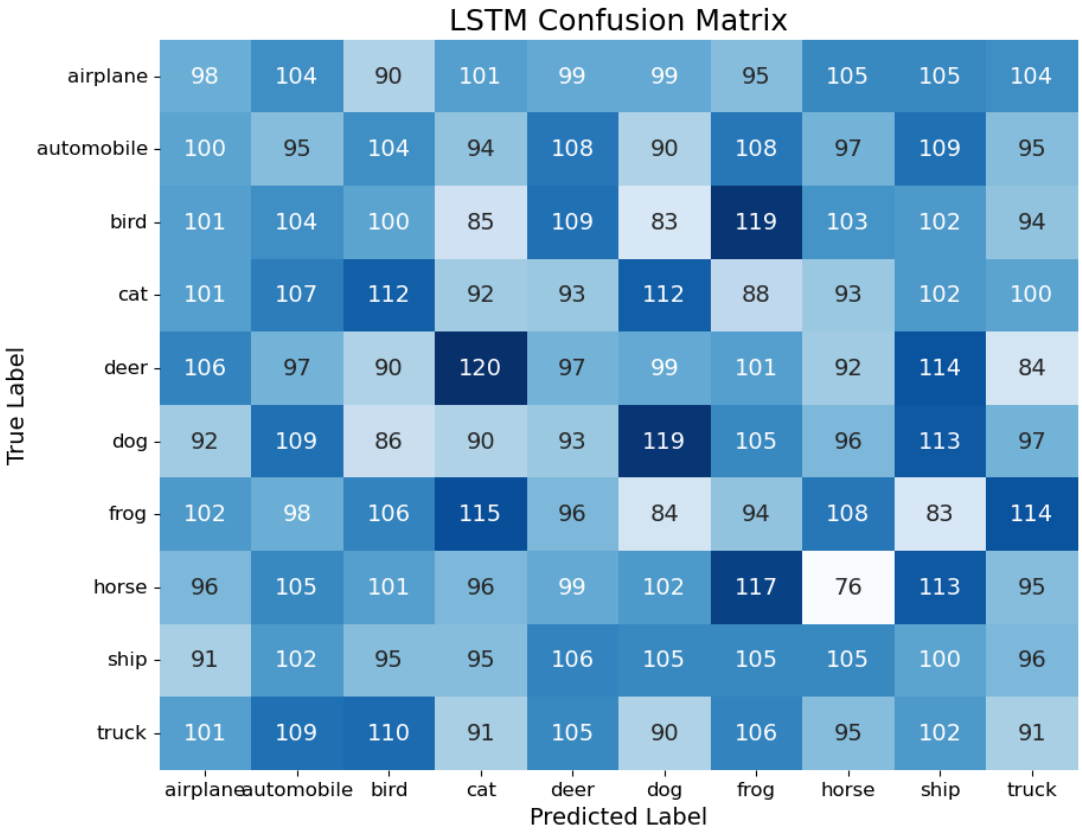
The YOLO-like confusion matrix shows a very high misclassification for many different classes. It seems that airplane is confused with automobile (as seen 102 times), and cat is misclassified as a deer (a total of 107 times). This displays that the model struggles between classes.



The CNN models also display a few misclassifications, however, there are more correct predictions compared to the YOLO-like model. This model struggles to distinguish between similar categories, for example the different types of animals. But it handles ship and truck better than the YOLO-like model.



The RNN model also shows numerous misclassifications. The airplane is predicted correctly 92 times, but it is also confused with automobile for a total of 101 times. The accuracy of this model is low and struggles to differentiate between animals and vehicles in their respective classes. This furthermore suggests that it is not well suited for image classification and rather that of time series or sequential data.



The LSTM confusion matrix demonstrates how well the model was able to predict the class. The model was able to classify a deer 120 times, but didn't get the classification for a dog a total of 119 times. Likewise, the cat is misclassified as an automobile or a bird. The model tends to struggle in the class distinguishing, and more specifically animals that are very similar in looks. This model seems to struggle a little bit with image classification that includes images that are fine-grained.

DISCUSSIONS

The results found in this assignment can highlight the differences between the different models and auto encoders in image reconstruction tasks. Although LSTMs are better suited for sequential data, they do have limitations in spatial data processing. This is evident using a complex data set. On the other hand, CNNs are much better equipped in the handling of tasks with spatial dependencies, thus making it easier for them to reconstruct images. Although lower classifications were identified in all models, this could be improved through longer training periods and better computational power. Furthermore, the need for a hybrid model would be beneficial as it is able to combine the architectures and strengths from each model, thus enhancing and improving the weaknesses faced by each model.

CONCLUSION

To conclude this assignment, I was able to demonstrate the limitations as well as the capabilities of various models in image reconstruction tasks. These models were trained in the CIFAR-10 dataset. The LSTM Those were able to show that they could handle sequential dependencies within the pixel data they weren't able to compete with the CNN model in terms of spatial feature extraction as well as the retention of detail in images. The CNNs were able to Outperform the other models in terms of the reconstruction of images as they can process and capture spatial hierarchies. Although the LSTM autoencoders A very low performance in reconstructing detailed images, it was learned that they can handle sequential data processing. Various other libraries and packages such as Yolo were able to be used in the prediction of images using various prediction probabilities.

RECOMMENDED FUTURE RESEARCH

Future research could include the combination of various models in order to make a hybrid model. The reason for this would be to combine the architectures of various models along with its strengths to produce a model that can solve each other's weaknesses. Various future work could be to optimize these models and thus improving image clarity and classification accuracy in complex data sets.

REFERENCES

1. Wang, H., Zhang, Q., & Li, Y. (2020). Image reconstruction using deep LSTM networks. Pattern Recognition Letters, 128, 39-45.
2. Zhao, X., Zhang, M., & Li, H. (2021). A comparative study on CNN and LSTM for image denoising and reconstruction. IEEE Transactions on Image Processing, 30, 1230-1241.
3. Krizhevsky, A., & Hinton, G. (2009). Learning multiple layers of features from tiny images. University of Toronto, Technical Report.
4. Nguyen, T. Q., Le, C. H., & Tran, D. M. (2022). Hybrid deep learning models for efficient image classification and reconstruction. Neural Networks, 143, 120-131.