MNIST Digits Image Classification and UAV Object Detection

Assignment – Deep Learning

Deepak Kovaichelvan - S336570

Date: 7th February 2022

Contents

1. MNIST Digits Image Classification

The objective of this section is to perform image classification on the MNIST handwritten digits dataset, samples of which are shown in Figure 1. This is a large dataset of handwritten digits that is widely used for training and testing image classification models in the field of machine learning.

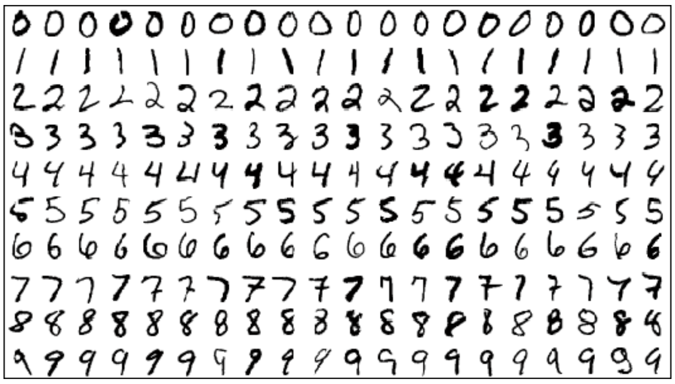


Figure 1: MNIST Digits Dataset [1]

1.1. Models

Two separate models were built to classify the handwritten digits – Fully Connected Layers and Convolutional Neural Network (CNN) using Python. The TensorFlow framework was used for creating the deep learning models. The aim is to beat the performance of the benchmark **LeNet 1998** model.

1.1.1. Fully Connected Layers

A fully connected (FC) neural network is made up of a series of layers that connect every neuron in one layer to every neuron in the following layer.

The fully connected neural network built for the digits classification is shown in Figure 2. The images in the MNIST digits dataset are 28 x 28 pixels. The images were pre-processed by reshaping into a 1D vector of length 784 to use as the input layer of the neural network, from which it would make predictions.

The architecture was developed experimentally, starting with a simple network and adding depth progressively to beat the LeNet 1998 model. The FC network is illustrated in Figure 2 and was constructed as follows:

1. The first hidden dense layer contains 256 neural units and uses a Rectified Linear Activation function (ReLU). The ReLU function captures the non-linearities in the data to allow for training on complex datasets.

2. The second, third and fourth hidden layers are dense layers, each containing 128, 64 and 32 units respectively and use the ReLU activation function.

3. The final layer is the output layer and contains 10 units, one for each of the digits. The layer uses the Softmax function to take inputs and convert into a probability distribution over the predicted output classes (digits).

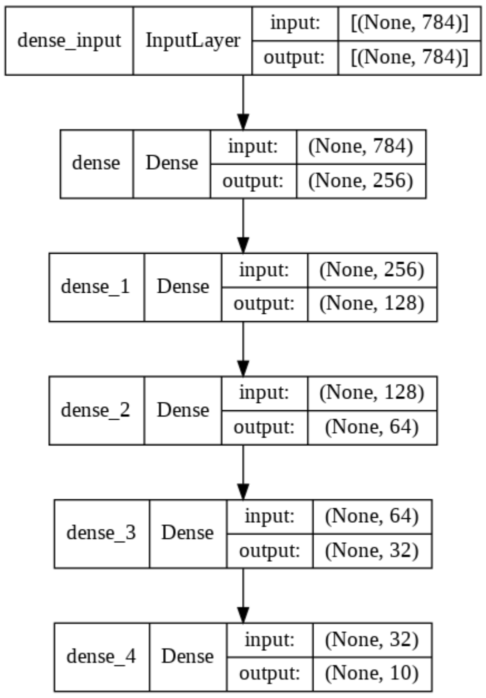


Figure 2: Fully connected neural network architecture

1.1.2. Convolutional Neural Network

CNN is one of the most important deep learning algorithms used in image detection. They employ four main types of layers to extract features – Convolutional, Pooling, Flatten and Fully Connected Layers. The convolutional and pooling layers are repeated multiple times to increase learning from abstract features [2].

The CNN is illustrated in Figure 3 and was constructed progressively with additional depth to beat the LeNet 1998 model as follows:

1. The first hidden layer is a convolutional layer with 32 units and uses the ReLU function. It accepts the images as 28 x 28 pixels from the input layer without reshaping. The convolution process works by moving 3 x 3 kernels across the receptive field of the image to check if a specific feature is present in the image [2].

2. The second layer is a max pooling layer. This process down samples the image using a 2 x 2 kernel to extract the maximum pixel value. By reducing the dimensions of the feature maps, they can reduce the number of parameters and computation required to process the image.

3. This was followed by the third and fourth hidden layers, which are both convolutional layers with 32 units, 3 x 3 kernel and ReLU activation function. These additional convolution layers help extract higher level features from the image to enable better detection capability.

4. The fifth hidden layer is a flatten layer, which converts the output from the fourth hidden layer into a 1D vector.

5. The sixth hidden layer is a dense layer with 32 units and ReLU activation function to which the 1D vector is passed.

6. The final layer is the output layer which uses Softmax function to come up with probability distribution over the predicted output classes to perform classification.

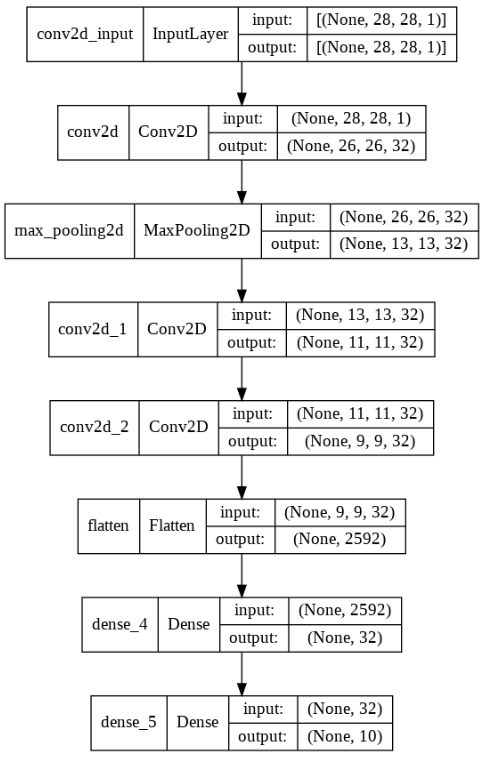

conv2d_input | InputLayer | input: [(None, 28, 28, 1)] | output: [(None, 28, 28, 1)]
conv2d | Conv2D | input: (None, 28, 28, 1) | output: (None, 26, 26, 32)
max_pooling2d | MaxPooling2D | input: (None, 26, 26, 32) | output: (None, 13, 13, 32)
conv2d_1 | Conv2D | input: (None, 13, 13, 32) | output: (None, 11, 11, 32)
conv2d_2 | Conv2D | input: (None, 11, 11, 32) | output: (None, 9, 9, 32)
flatten | Flatten | input: (None, 9, 9, 32) | output: (None, 2592)
dense_4 | Dense | input: (None, 2592) | output: (None, 32)
dense_5 | Dense | input: (None, 32) | output: (None, 10)

Figure 3: CNN architecture

1.2. Hyperparameters

The following hyperparameters were tuned in the deep learning models:

1. Epochs is the number of times the learning algorithm works through the dataset through forward and backpropagation processes. The number of epochs was set to a high enough number so that the model could learn from the training data.

2. Batch size is an important hyperparameter affecting the training process. It is the number of samples from the training dataset processed before the model is updated. Small batch sizes guarantee convergence, at the cost of high computational time. Large batch size results in faster computational, at the cost of poor generalisation [3].

3. Early stopping was used to monitor the loss-function with number of epochs. The training loss decreases with increasing number of epochs, but beyond a certain limit the model fits to the noise in the training data and cannot generalise to new data. Therefore, the loss function of the hold-out validation dataset is monitored with the loss function of the training dataset. When there is an increase in validation loss at an epoch when compared to previous epochs, the learning process can be stopped by specifying a patience parameter.

4. Learning rate is a crucial hyperparameter which decides the extent to which the new weights are changed after each epoch. Specifying too large learning rates will lead to unstable learning process and potentially miss the minima. In contrast, small learning rate will lead to long computation time. The best approach is to use an adaptive learning rate as shown in Figure 4. This was implemented by using a proven optimisation algorithm, Adaptive moment estimation (Adam), with a specified initial learning rate [2].

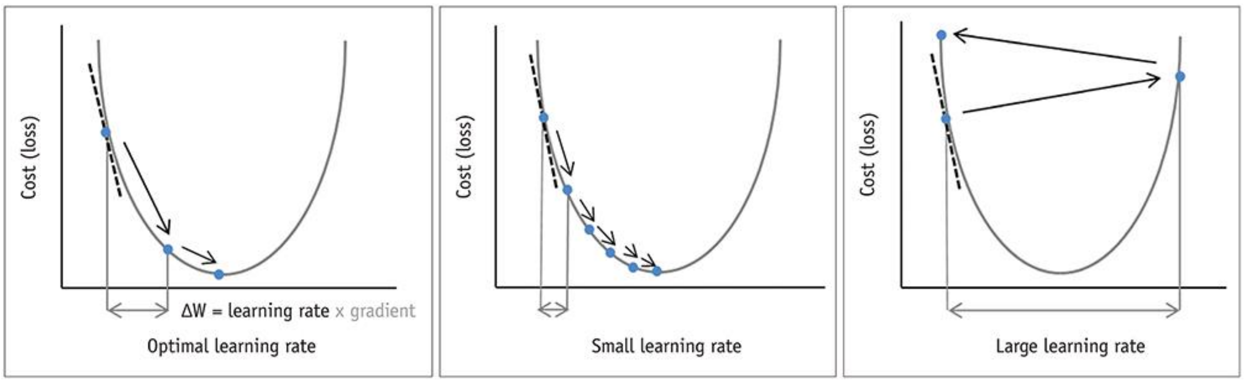


Figure 4: Effect of learning rate on weight optimisation [4]

The hyperparameters were first trialled through an experimental process, to study the effects on the predictions and optimise. The final hyperparameters for the two networks are summarised in Table 1.

Table 1: Summary of hyperparameters used in the FC and CNN models

	Fully Connected Layers	**CNN**
Epochs	75	75
Batch Size	512	256
Learning Rate Optimiser	Adam	Adam
Initial Learning Rate	0.0001	0.0001
Early Stopping Patience	3 epochs	3 epochs
Validation Split	20%	20%

1.3. Monitoring the Model Training

The training of the neural networks is monitored by plotting the cross-entropy loss, which increases when the predicted probability diverges from the actual label. Additionally, the accuracy of predictions was plotted on the training dataset and a holdout validation dataset against the number of epochs.

The MNIST training dataset contains 60,000 images, of which 10,000 (20%) images were set aside for the holdout validation set and the remaining 50,000 images were used for training.

The learning curve from the training dataset shows how well the model is learning, while the learning curve from the validation dataset shows how well the model is able to generalise. Training the model beyond a certain number of epochs will result in the overfitting of the model to the noise in the training dataset and will no longer be able to generalise on an unseen dataset. This is the motivation for monitoring the training process.

Figure 5 and Figure 6 show the learning curves for the Fully Connected Layers and CNN models respectively. Early stopping monitor was implemented with a patience value of 3 epochs. When the validation loss increases beyond 3 consecutive epochs, the training stops and the best weights before the trigger is applied.

The figures show that both the models have achieved good fit. This means that the validation loss and accuracy have both reached a level of **stability** and have a **minimal gap** with the training curves. Further training will result in an overfit model. The plots show that the FC network completed training at 47 epochs, while the CNN completed training at 16 epochs.

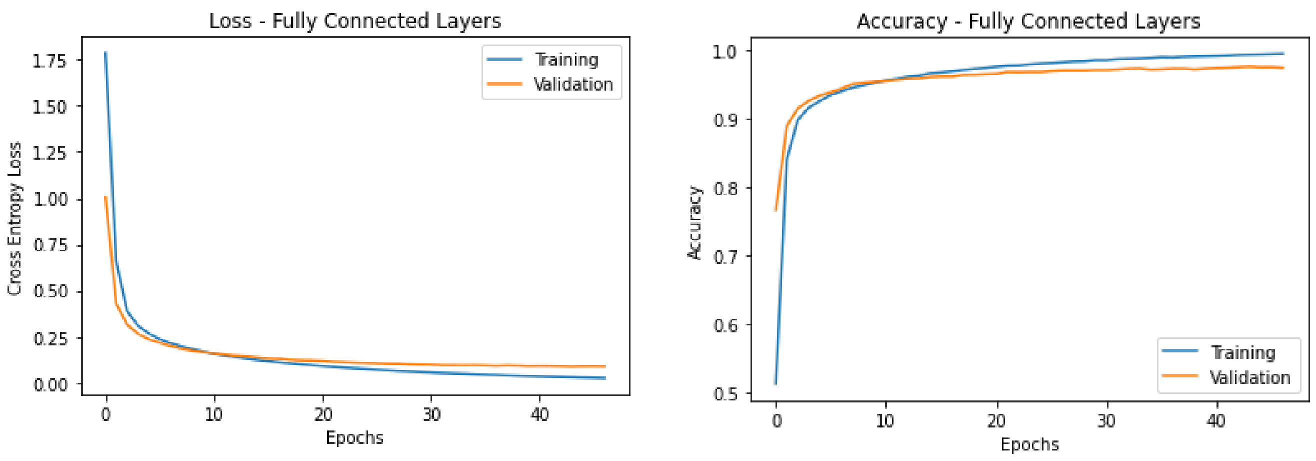


Figure 5: Learning curves for Fully Connected Layers with respect to number of epochs

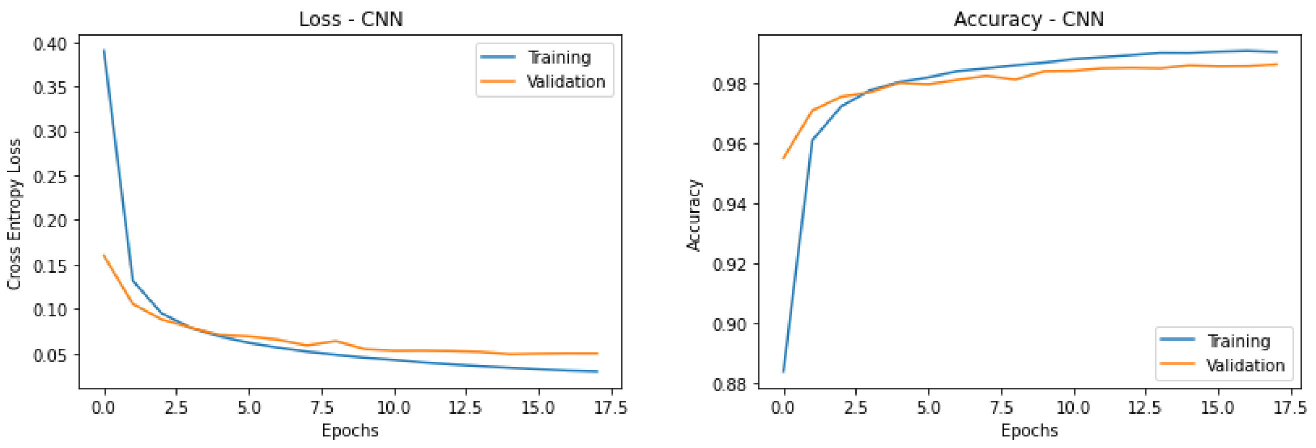


Figure 6: Learning curves for CNN with respect to number of epochs

1.4. Performance on the Test Dataset

The trained models from the FC network and CNN were then evaluated on the test dataset of 10,000 images, which is an unseen dataset. The accuracy of predictions of the two models on the test dataset is summarised in Table 2.

Both the models achieve a high level of accuracy, with the CNN performing better in this case. The performance of the CNN beats the LeNet 1998 model, which achieved an accuracy of 98.7% accuracy. The FC network matches the accuracy of the LeNet 1998 model.

This was achieved without the model overfitting the training data, as already discussed in section 1.2 through the validation loss monitoring.

The main reason that the CNN performs better than the FC network is that it is specifically tailored to extract features from images using convolutional layers and kernels. In addition, CNN has fewer parameters to train, making it computationally more efficient.

Table 2: Summary of model performance on the test dataset

	Test Accuracy
Fully Connected Layers	97.68%
Convolutional Neural Network	98.9%
Benchmark – **LeNet 1998**	98.7%

2. UAV Object Detection using Transfer Learning

The dataset provided for this part of the assignment contains images of 314 UAVs of different types, in different environments and zoom settings. Each image has a corresponding text file containing the information on the localisation of the UAV in the image, this was already done by manual labelling and will be treated as the ground truth bounding boxes (BB). The objective is to train a model to predict the localisation of the drones and verify the accuracy of the predictions against the ground truth.

2.1. Data Analysis and Pre-Processing

2.1.1. Imported Dataset

The image dataset was imported into Python. Figure 7 shows some example images from the dataset.

shutterstock.com · 724434283
shutterstock.com · 726603313
shutterstock.com · 710602144

Figure 7: Example images from the UAV images dataset

The text files containing image labels were imported into Python. The format of the information in the text files is explained in Figure 8.

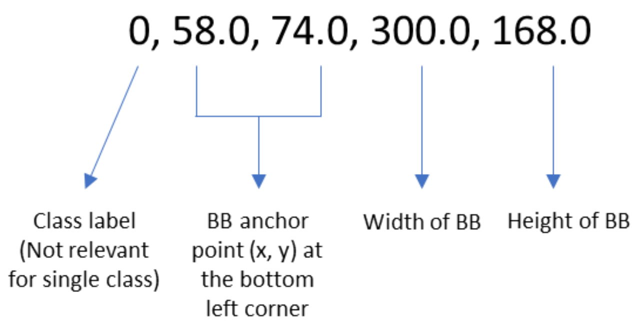

0, 58.0, 74.0, 300.0, 168.0
Class label (Not relevant for single class)
BB anchor point (x, y) at the bottom left corner
Width of BB
Height of BB

Figure 8: Format of the image labels (bounding boxes) dataset

2.1.2. Verification of Label Format

The label format was verified by plotting the ground truth bounding boxes on the images using the Matplotlib library and ensured that they correctly capture the localisation of the UAVs.

Figure 9 shows the ground truth bounding boxes overlaid on the images. It can be seen that the localisation of the UAVs is correctly captured, thereby verifying the format of the labels.

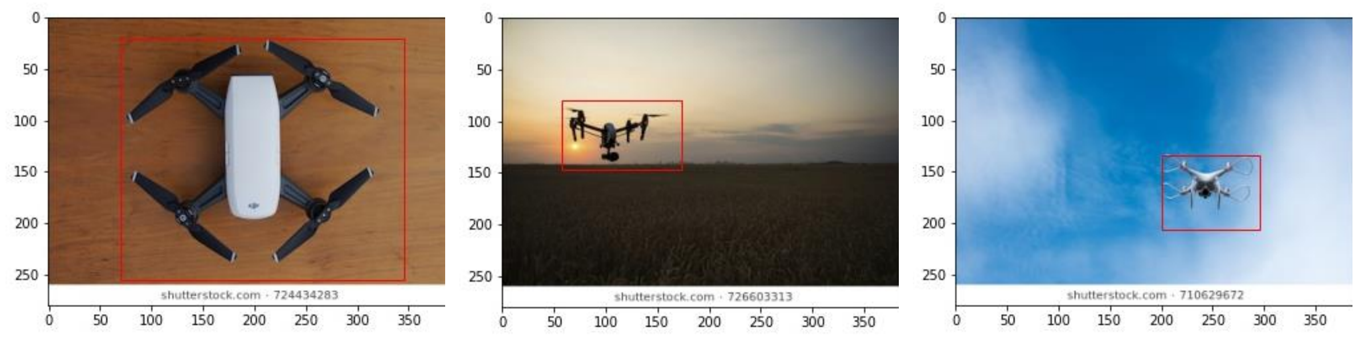

shutterstock.com · 724434283
shutterstock.com · 726603313
shutterstock.com · 710629672

Figure 9: Verification of label format by plotting the ground truth labels

2.1.3. Pre-Processing the Dataset

Prior to training the models, the images were pre-processed to prepare them for the pre-trained network. In this case, VGG16 was selected as the base network for the transfer learning model.

Data Cleaning
It was noticed that some of the bounding box labels were inaccurate and did not correctly capture the localisation of the UAVs, which could lead to poor model performance. As a result, 4 images of the most inaccurate labels were removed from the dataset, bringing the size of the dataset to 310 images.

Image Pre-Processing
The images were pre-processed as follows:
1. Images were loaded and resized to 224 x 224 pixels, as required by the VGG16 model.
2. The images were converted to NumPy arrays so that they can work with TensorFlow.
3. The arrays contain pixel intensities in the range [0, 255]. This was normalised to the scale [0, 1] by dividing all the values by 255.

The final output is a list of arrays, each of shape (224, 224, 3) containing the pixel intensities of the 3 channels (RGB) of the images normalised in the range [0, 1].

Label Pre-Processing
The bounding boxes were pre-processed as follows:
1. The height and width of the images were extracted from each of the images.
2. The bounding box dimensions were scaled with respect to the height and width of the image, in the range [0, 1].

Train-Test Split
The pre-processed images are the input (X) to the model and the pre-processed labels are the truth labels (y_true) against which the predicted labels (y_pred) will be compared.

Due to the availability of a relatively small dataset of 310 images after data cleaning, it would be beneficial to use the maximum number of images for training in order to generate a well performing model, while still having a diverse test set to evaluate performance on.

Considering this, the dataset was divided into train and test sets using a random split as follows:
1. 260 images for training (of which 15% was set aside as the holdout validation set)
2. 50 images for testing

2.2. Object Detection Model

2.2.1. Model Architecture

The bounding box regressor was constructed by tailoring the pre-trained VGG16 network for a regression problem as follows [5]:

1. The pre-processed images in the array form of shape (224, 224, 3) make up the input layer of the model.

2. The VGG16 network with pre-trained ImageNet weights make up the initial hidden layers. These layers are comprised of a combination of convolutional and max pooling layers to extract features from the image. The final layer in the VGG16 network is a Softmax layer by default, however since the objective is to perform regression, the layer was excluded in this network.

3. Customised layers were added hereon to re-purpose the VGG16 for regression. The output from the VGG16 network was flattened into a 1D vector with a flatten layer, which is the first custom layer.

4. The following 3 layers are hidden layers utilising the ReLU activation function.

5. The output layer has 4 units and uses a Sigmoid activation function, which will provide 4 values, each representing the bounding box coordinates for plotting. The Sigmoid function is used as the bounding boxes are scaled to the image size, therefore needs to be in the range [0, 1].

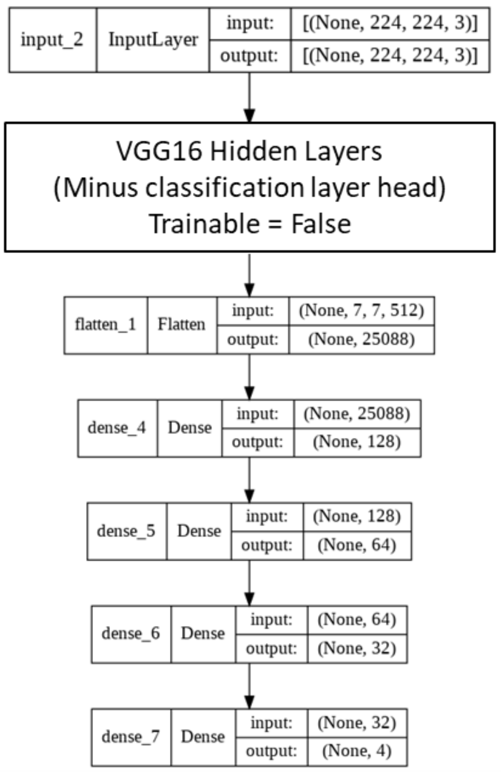

input_2 | InputLayer | input: [(None, 224, 224, 3)] | output: [(None, 224, 224, 3)]
VGG16 Hidden Layers
(Minus classification layer head)
Trainable = False
flatten_1 | Flatten | input: (None, 7, 7, 512) | output: (None, 25088)
dense_4 | Dense | input: (None, 25088) | output: (None, 128)
dense_5 | Dense | input: (None, 128) | output: (None, 64)
dense_6 | Dense | input: (None, 64) | output: (None, 32)
dense_7 | Dense | input: (None, 32) | output: (None, 4)

Figure 10: Bounding box regressor with pre-trained VGG16 network

2.2.2. Hyperparameters

Through a process of experimentation, different hyperparameters were trialled to study the effects on the model performance. The final hyperparameters used in the model are summarised in Table 3. The significance of these hyperparameters has been explained in section 1.1.3.

Table 3: Hyperparameters of the VGG16 based bounding box regressor

	VGG16 BB Regressor
Epochs	60
Batch Size	32
Learning Rate Optimiser	Adam
Initial Learning Rate	0.0001
Early Stopping Patience	4 epochs
Validation Split	15% of training set

2.2.3. Monitoring the Model Training

The learning process was monitored by plotting the loss function against the number of epochs. The loss function was plotted on the training and validation sets.

Mean squared error (MSE) was chosen as the loss function, since this is an effective measure of the mean difference between the true and predicted values in regression problems. The model was fit to the training data with an epoch number of 60 to allow for the model to learn from the training data.

An early stopping monitor callback with a patience of 4 epochs was used to monitor and stop the training process when the validation loss increases beyond the allowed number of epochs. The best weights are restored. This allows the model to learn from the training data without overfitting. The early stopping monitor ensures that the model is able to generalise on unseen data.

The learning curve is shown in Figure 11. It can be seen that the model has completed training at 34 epochs, as it was stopped by the early stopping callback. The model has achieved a good fit as the losses have reached a level of stability with a minimal gap.

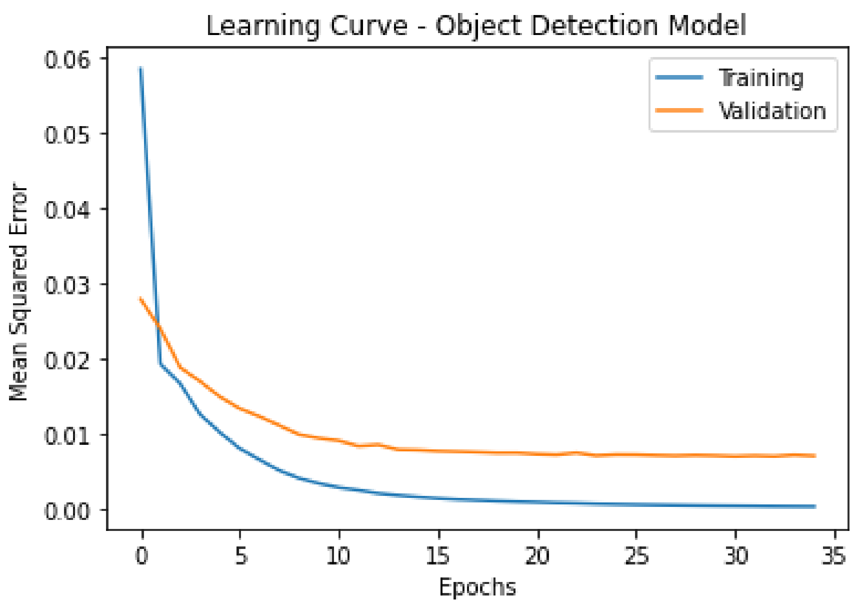


Figure 11: Learning curve showing the mean squared error on the training and validation sets versus epochs

2.2.4. Performance on the Test Dataset

The performance of the model has been illustrated using a few of the test images in Table 4. The predicted bounding boxes are plotted in green and ground truth bounding boxes are plotted in red.

It can be seen that the model achieves varying levels of accuracy on different test images. There are a few images where the truth and predictions overlap almost completely with each other, while there are a few other images, where the overlap has deteriorated. On a small number of the test images, the predictions almost completely miss the targets.

Intersection over Union (IoU) is a widely used evaluation metric to measure the accuracy of object detectors. This metric requires the ground truth and predicted bounding boxes, which are both known at this stage. The IoU can be computed for any pair of bounding boxes on an image as illustrated in Figure 12. IoU is simply the area of overlap between the ground truth and predicted bounding boxes, over the area of the union of the two bounding boxes [6].

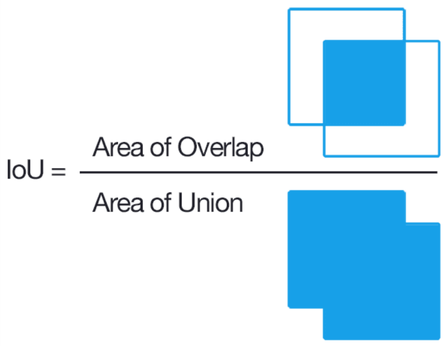


Figure 12: Computing the Intersection over Union [6]

The IoU for each image from the test dataset was collected in a list to study the object detector performance. The variation in IoU performance on the test images is depicted in Figure 13. It can be seen that the IoU performance has a left-skewed normal distribution, with an average IoU across all test images of **0.635**. The highest IoU distribution is around the range 0.65-0.9. At the tail of the distribution, there are a small number of image predictions that achieved IoU in the range 0-0.4.

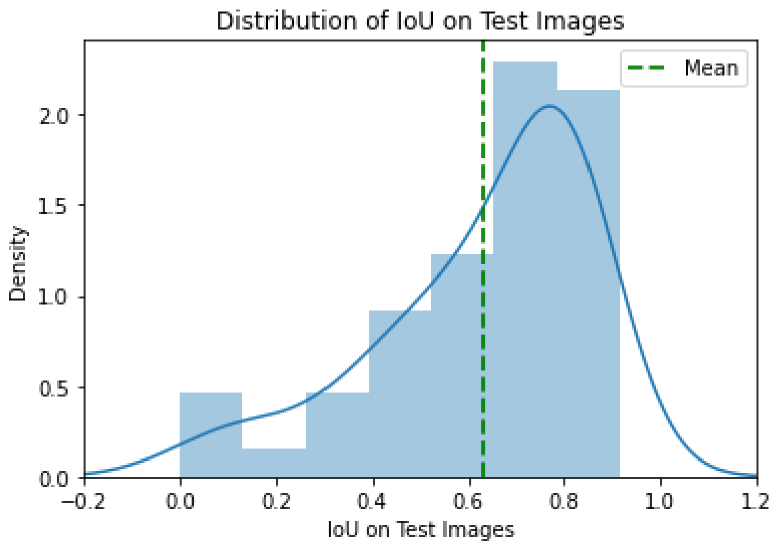
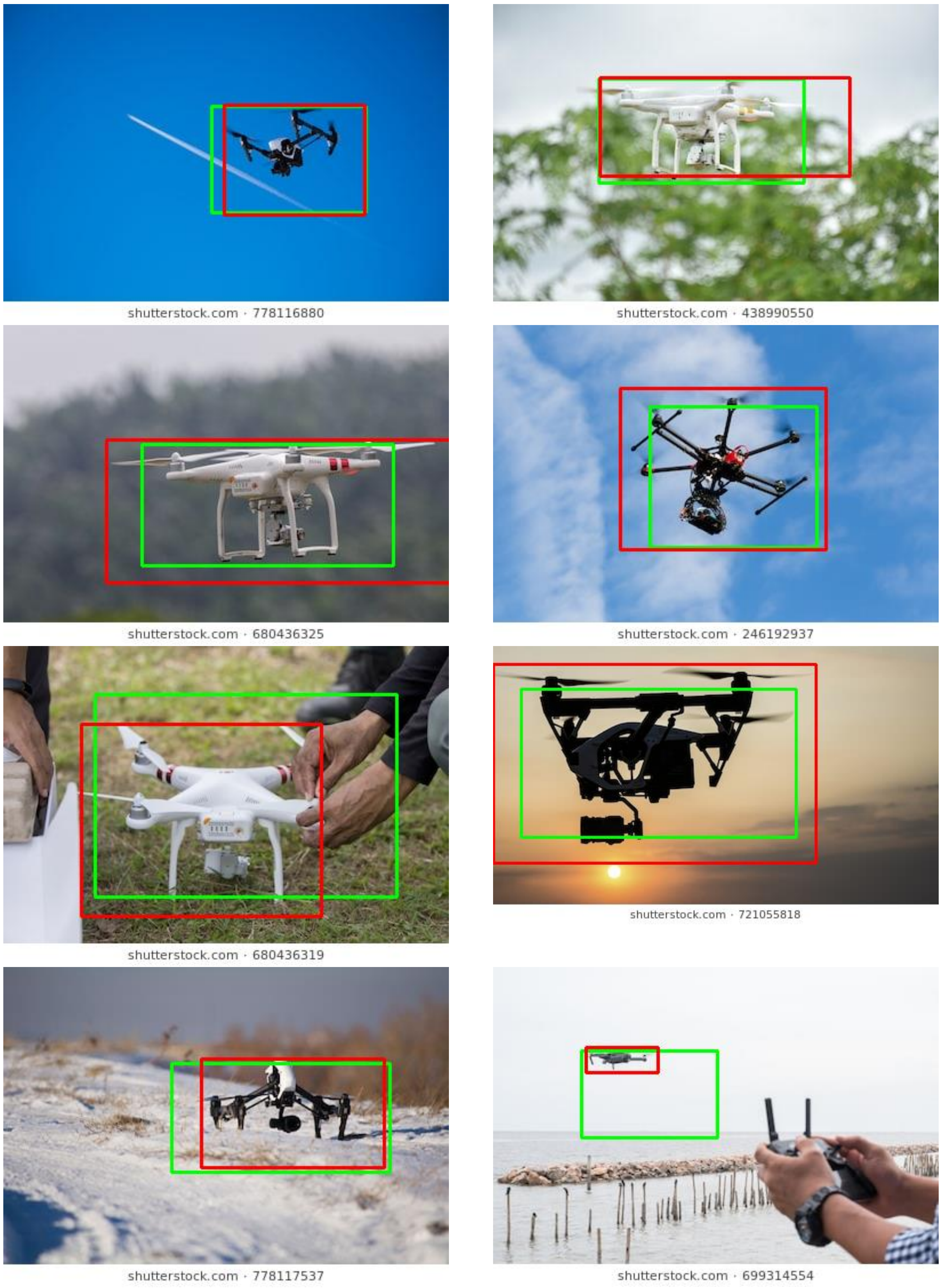
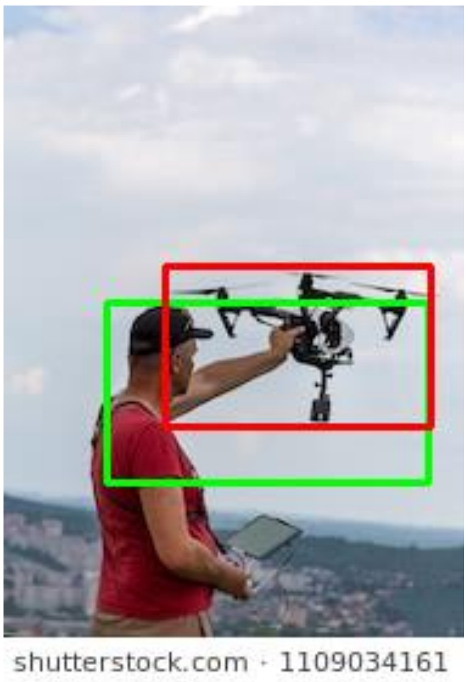
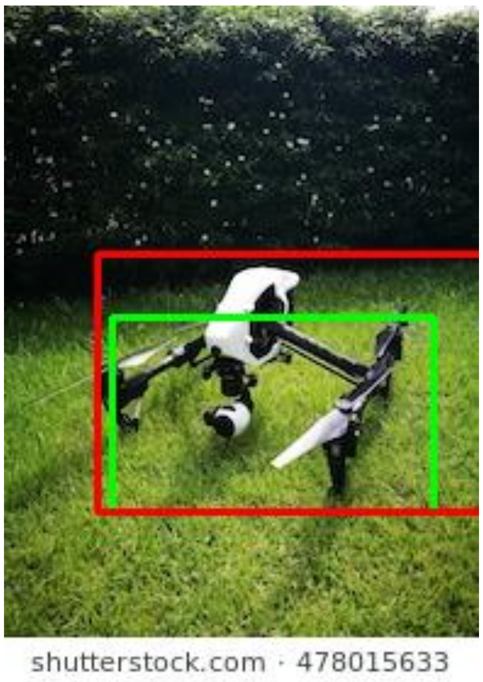


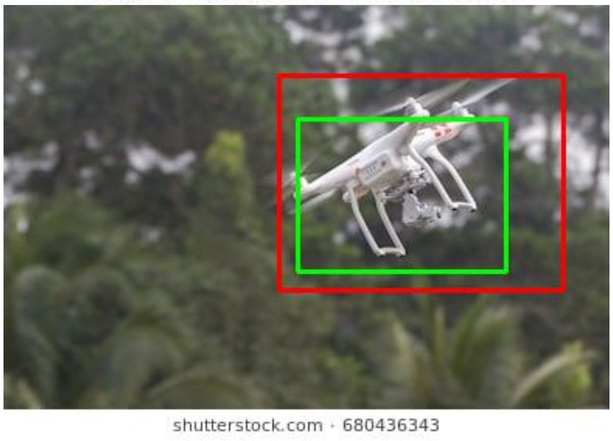
Figure 13: Distribution plot depicting the variation in IoU on test images

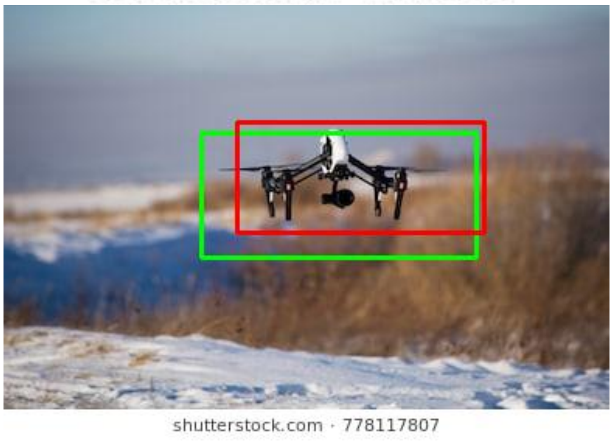
Table 4: Plots of predicted (green) and ground truth (red) bounding boxes on the test dataset


shutterstock.com · 778116880
shutterstock.com · 438990550
shutterstock.com · 680436325
shutterstock.com · 246192937
shutterstock.com · 680436319
shutterstock.com · 721055818
shutterstock.com · 778117537
shutterstock.com · 699314554









2.2.5. Factors Limiting the Performance

The performance of the object detection regressor is limited due to the below factors and can be addressed as follows:

1. Relatively small size of the dataset is a limiting factor. Increasing the size of the dataset to a few thousand images can significantly improve the IoU performance. This will provide more diversity in the data and allow for better training of the object detection regressor.

2. Although some of the poorly labelled images were removed, the labelling of the remaining dataset is still not accurate. The model not having the accurate ground truth negatively influences the performance of the training and predictions. Therefore, relabelling the images will improve the performance.

Apart from the above-mentioned measures, further work can be done to experiment different pre-trained models (such as ResNet101) and hyperparameters (such as batch size, patience) and the depth of the model by adding custom layers and neural units. These steps will result in improved performance of the model.

3. References

[1] "MNIST Using Recurrent Neural Network | by Ting-Hao Chen | Machine Learning Notes | Medium." https://medium.com/machine-learning-algorithms/mnist-using-recurrent-neural-network-2d070a5915a2 (accessed Feb. 03, 2022).

[2] S. Albahli, F. Alhassan, W. Albattah, and R. U. Khan, "Handwritten Digit Recognition: Hyperparameters-Based Analysis", doi: 10.3390/app10175988.

[3] J. Brownlee, "How to Control the Stability of Training Neural Networks With the Batch Size," 2019. https://machinelearningmastery.com/how-to-control-the-speed-and-stability-of-training-neural-networks-with-gradient-descent-batch-size/ (accessed Jan. 28, 2022).

[4] "How to Use Learning Rate Annealing with Neural Networks?" https://analyticsindiamag.com/how-to-use-learning-rate-annealing-with-neural-networks/ (accessed Jan. 28, 2022).

[5] "Object detection: Bounding box regression with Keras, TensorFlow, and Deep Learning - PyImageSearch." https://www.pyimagesearch.com/2020/10/05/object-detection-bounding-box-regression-with-keras-tensorflow-and-deep-learning/ (accessed Feb. 02, 2022).

[6] "Intersection over Union (IoU) for object detection - PyImageSearch." https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/ (accessed Feb. 03, 2022).

Appendices

Appendix A – Python Code for MNIST Digits Classification

The MNIST digits classification code is available to download from the below URL:
MNIST Digits Classification

Appendix B – Python Code for UAV Object Detection

The UAV object detection code is available to download from the below URL:
UAV Object Detection