Deep learning

Assignment

Quentin Fever - 383387

Department: Applied Artificial Intelligence SATM

Date: 10/02/2023

All the code related to this assignment is available following this github link:
https://github.com/quent1fvr/Drone_detection

Table of Contents

Task 1: Image classification on the MNIST handwritten digits recognition dataset

Objectives

The MNIST dataset is a widely used dataset in handwritten digit recognition. This image classification task is considered a standard classification problem for neural networks. In this report, we will examine the different methods used to solve the handwritten digit recognition task on the MNIST dataset. In particular, we will investigate the use of fully connected layers and CNNs for handwritten digit recognition. Furthermore, we will evaluate the performance of these methods in terms of accuracy and speed. Finally, we will discuss the transfer learning method that can be used to improve the performance of neural networks on the MNIST dataset.

Figure 1: Image example from MNIST dataset

Pre-Processing

A crucial aspect of preparing the data for the MNIST dataset is normalizing the pixel values. The initial pixel values for each image range from 0 to 255 and are represented as integers. To standardize these images, it is necessary to rescale the pixel values of both the training and test sets to fall within a range of 0 to 1. This is achieved by dividing each pixel value by 255, the highest possible value for a pixel.

1.a) Fully connected layers and CNN for handwritten recognition

Fully connected network

A fully connected neural network (FC) is a classic type of neural network that is densely connected, i.e. each neuron in one layer is connected to all neurons in the next layer. It is called "fully connected" because all neurons are connected to each other. CF treats each input as a single set, which means that it does not take into account the spatial structure of the input data.

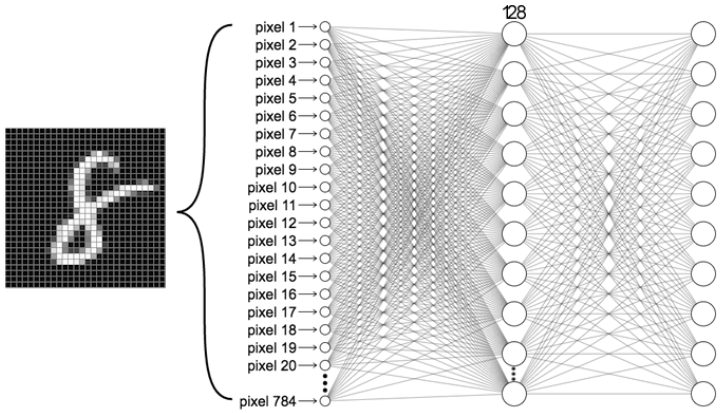


Figure 2: visualization of a fully connected model

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense (Dense)               (None, 1024)              803840

 dense_1 (Dense)             (None, 512)               524800

 dense_2 (Dense)             (None, 200)               102600

 dense_3 (Dense)             (None, 150)               30150

 dense_4 (Dense)             (None, 10)                1510

=================================================================
Total params: 1,462,900
Trainable params: 1,462,900
Non-trainable params: 0
```

Figure 3: Fully connected neural network for training

A sequential model is initialized with a Flatten layer and five fully connected layers with the ReLU activation function. The layers have respectively 1024,512,200,150 and 10 neurons with the Softmax activation function at the end to convert the output into a probability distribution over the classes. The number of neurons in the final layer is 10, matching the number of classes in the MNIST dataset.

Convolutional neural network

A convolutional neural network (CNN) is a type of neural network dedicated to image recognition. It uses convolutional filters to extract features from the spatial structure of the input data. Instead of treating each input as a single set, the CNN uses convolutional layers to capture spatial relationships in the data. The convolution layers are followed by max-pooling layers that reduce the dimensionality of the data to facilitate generalization and prevent overlearning. Finally, the CNN uses FC layers to perform image classification.

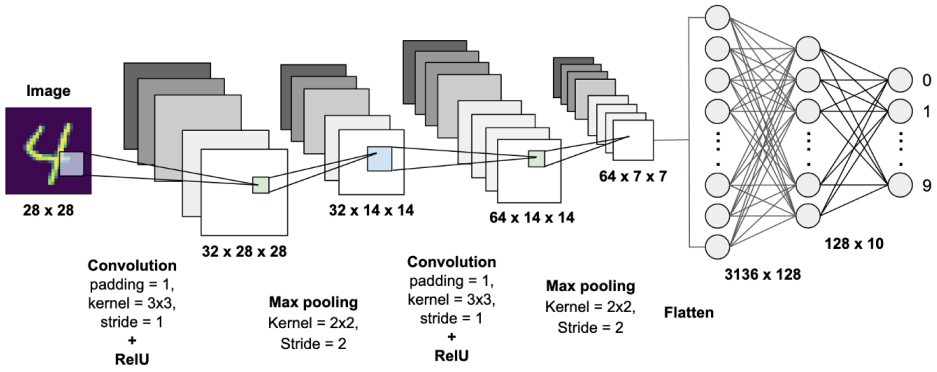
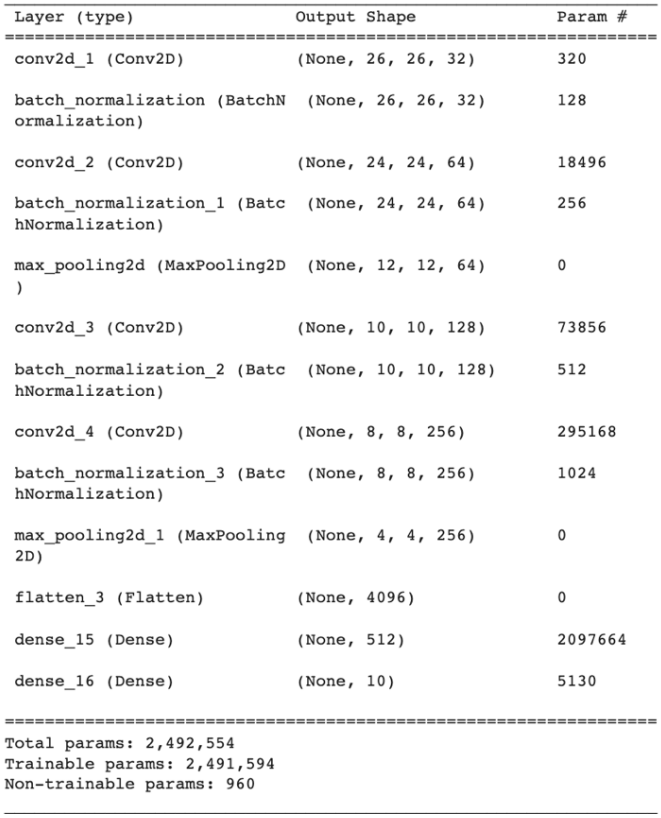


Figure 4: visualization of the convolutional neural network model (https://towardsdatascience.com/mnist-handwritten-digits-classification-using-a-convolutional-neural-network-cnn-af5fafbc35e9)

```
model2 = Sequential()

model2.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(28, 28, 1)))
model2.add(BatchNormalization())
model2.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model2.add(BatchNormalization())
model2.add(MaxPooling2D(pool_size=(2, 2)))

model2.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model2.add(BatchNormalization())
model2.add(Conv2D(256, kernel_size=(3, 3), activation='relu'))
model2.add(BatchNormalization())
model2.add(MaxPooling2D(pool_size=(2, 2)))

model2.add(Flatten())
model2.add(Dense(512, activation='relu'))
model2.add(Dense(10, activation='softmax'))

model2.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
# Train Model B
history2 = model2.fit(train_images , train_labels, epochs=20, batch_size=32,validation_data=(val_images, val_labels))
```

Figure 5: Code associated with the model



Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 26, 26, 32)	320
batch_normalization (BatchNormalization)	(None, 26, 26, 32)	128
conv2d_2 (Conv2D)	(None, 24, 24, 64)	18496
batch_normalization_1 (BatchNormalization)	(None, 24, 24, 64)	256
max_pooling2d (MaxPooling2D)	(None, 12, 12, 64)	0
conv2d_3 (Conv2D)	(None, 10, 10, 128)	73856
batch_normalization_2 (BatchNormalization)	(None, 10, 10, 128)	512
conv2d_4 (Conv2D)	(None, 8, 8, 256)	295168
batch_normalization_3 (BatchNormalization)	(None, 8, 8, 256)	1024
max_pooling2d_1 (MaxPooling2D)	(None, 4, 4, 256)	0
flatten_3 (Flatten)	(None, 4096)	0
dense_15 (Dense)	(None, 512)	2097664
dense_16 (Dense)	(None, 10)	5130

Total params: 2,492,554
Trainable params: 2,491,594
Non-trainable params: 960

Figure 6: CNN model summary

This model is a convolutional neural network to classify images from the MNIST dataset. It consists of a series of convolution layers followed by batch normalization and max pooling layers, which indicate the dimension of the image as features are extracted. The output of the convolutional and max pooling couches is then flattened and sent to fully connected couches to produce classification scores for the 10 classes of the MNIST. Batch normalization layers adjust the input data to avoid issues of unit saturation and oscillating activation distribution over time.

Feature Maps

Visualizing the Feature Maps of a convolution network can be very useful for understanding how a model processes visual information. The intermediate layers of the convolution network, called Feature Maps, produce intermediate representations of the input data. By visualizing the activations of the Feature Maps, we can observe how the convolution network identifies and extracts important features from the input.

This can help solve problems such as misclassification by showing which regions of the input were most influential in making a decision. It can also help understand how a model learned to generalize to validation data by visualizing similar regions of inputs that were activated for similar classes.

In addition, visualizing the Feature Maps can also give us insights into the limitations of our model and where we can improve its performance. For example, if we observe that the model is not paying attention to certain regions of the input, we can add layers to improve the model's ability to capture that information.

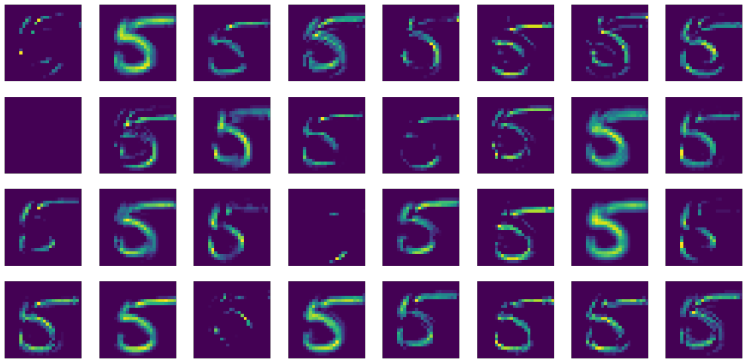


Figure 7: Feature map: first convolutional layer for the first input image

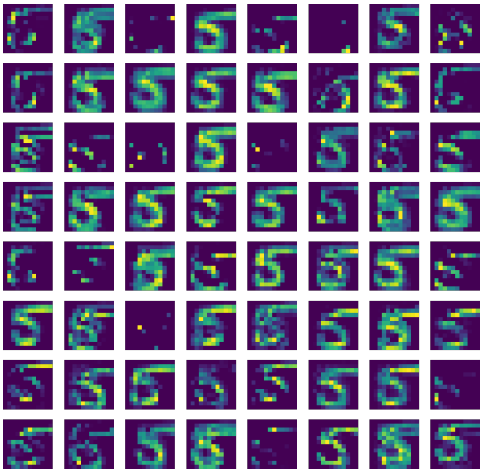


Figure 8: Feature map: last convolutional layer for the first input image

`

1.b) Performances

Model Fully connected

The final epoch resulted in an accuracy of 99.15% which is highly impressive (as shown in Figure 4). The neural network demonstrated a classification accuracy of 98% for the validation data. When it came to the unseen data, it achieved an accuracy of 98.75%.

Model CNN

The final epoch resulted in an accuracy of 99.76% which is highly impressive (as shown in Figure 4). The neural network demonstrated a classification accuracy of 99% for the validation data. When it came to the unseen data, it achieved an accuracy of 98.95%.

Learning curves

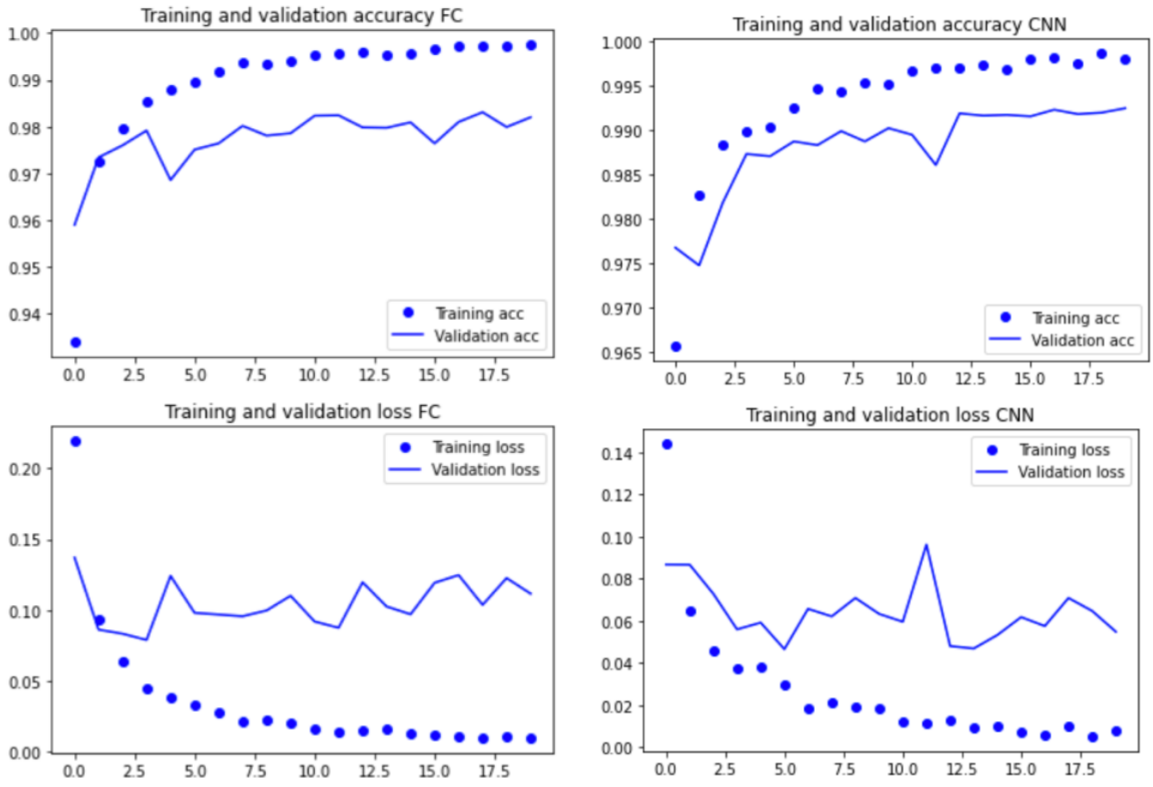


Figure 9: Learning curves of both models

In both models, the learning curve of the validation model follows the training curve, which may justify that the model generalises well on new data. When the loss curve for the training data continues to decrease over time, but the loss curve for the validation data begins to increase, this may indicate overfitting, as the model begins to overfit to the training data at the expense of generalizing to the validation data. This is not the case for the two trained models here.

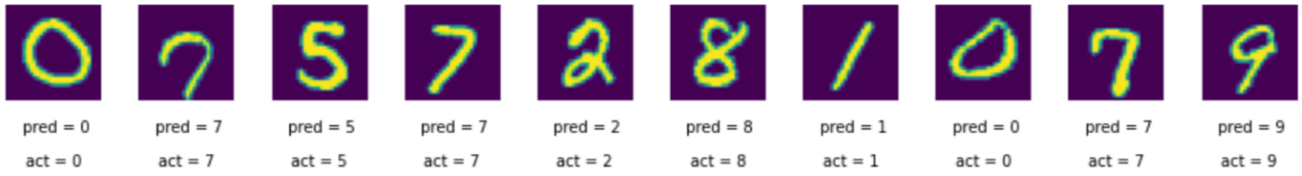


Figure10: Example of random images prediction

Because of the simplicity, both models manage to provide similar and satisfactory results. It may seem overkill to develop such complex models on data like MNIST

1.c) Transfer learning method

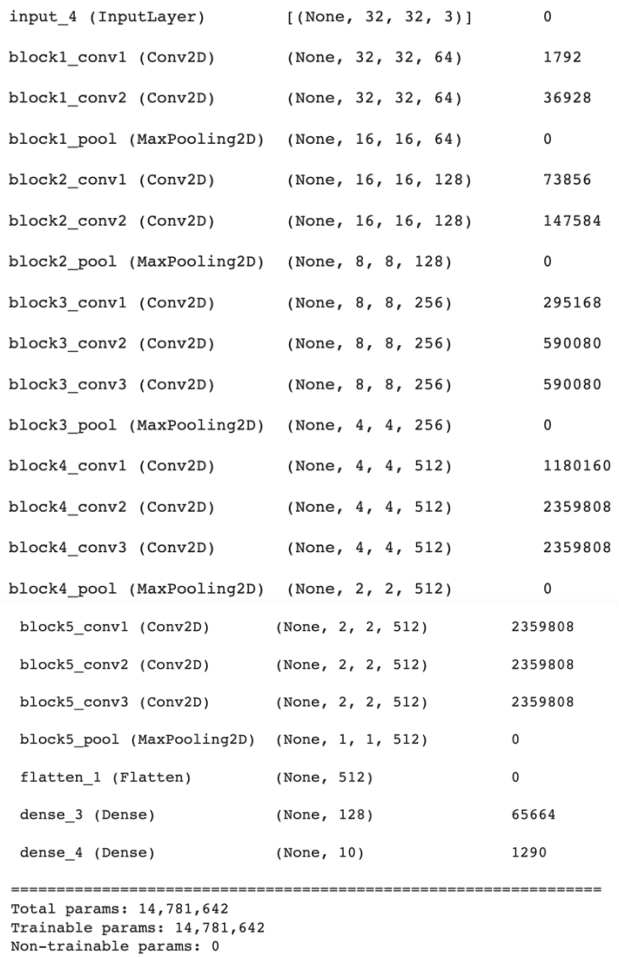
VGG16 model

In this section, we will use the VGG16 model and apply the transfer learning method on the MNIST dataset. The goal is to improve the performance of the model by using the pre-trained weights of VGG16 on other similar tasks. To do this, the MNIST images were resized to match the input dimensions expected by the VGG16 model. By then training this model on the MNIST data, it is hoped to obtain results superior to those obtained with a model trained from scratch.

VGG16 is a pre-trained convolutional image recognition model developed by the Visual Geometry Group research team at Oxford University. It was published in 2014 as part of the ImageNet Large Scale Visual Recognition Challenge and has since become a benchmark model in image recognition.
VGG16 consists of several convolutional layers, fully-connected layers, and data normalization layers. The structure of VGG16 is based on an architecture of small convolutional layer blocks repeated several times

In the case of MNIST with VGG, we can use a VGG model pre-trained on an image recognition task, such as object recognition, and reuse it to solve the handwritten digit recognition task. Instead of training the model on zero for handwritten digit recognition, we can use the pre-trained weights and adjust them to fit our new task.

This technique avoids over-training the model on the first task, which is a common problem when there is little data available for the new task. Transfer learning also makes efficient use of the data and resources already invested in training the previous model.



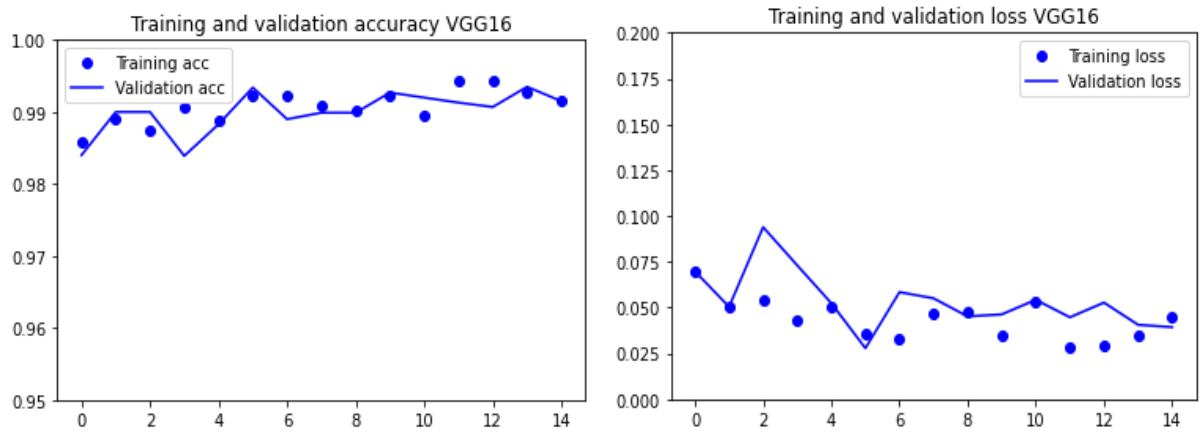
input_4 (InputLayer)	[(None, 32, 32, 3)]	0
block1_conv1 (Conv2D)	(None, 32, 32, 64)	1792
block1_conv2 (Conv2D)	(None, 32, 32, 64)	36928
block1_pool (MaxPooling2D)	(None, 16, 16, 64)	0
block2_conv1 (Conv2D)	(None, 16, 16, 128)	73856
block2_conv2 (Conv2D)	(None, 16, 16, 128)	147584
block2_pool (MaxPooling2D)	(None, 8, 8, 128)	0
block3_conv1 (Conv2D)	(None, 8, 8, 256)	295168
block3_conv2 (Conv2D)	(None, 8, 8, 256)	590080
block3_conv3 (Conv2D)	(None, 8, 8, 256)	590080
block3_pool (MaxPooling2D)	(None, 4, 4, 256)	0
block4_conv1 (Conv2D)	(None, 4, 4, 512)	1180160
block4_conv2 (Conv2D)	(None, 4, 4, 512)	2359808
block4_conv3 (Conv2D)	(None, 4, 4, 512)	2359808
block4_pool (MaxPooling2D)	(None, 2, 2, 512)	0
block5_conv1 (Conv2D)	(None, 2, 2, 512)	2359808
block5_conv2 (Conv2D)	(None, 2, 2, 512)	2359808
block5_conv3 (Conv2D)	(None, 2, 2, 512)	2359808
block5_pool (MaxPooling2D)	(None, 1, 1, 512)	0
flatten_1 (Flatten)	(None, 512)	0
dense_3 (Dense)	(None, 128)	65664
dense_4 (Dense)	(None, 10)	1290

```
=================================================================
Total params: 14,781,642
Trainable params: 14,781,642
Non-trainable params: 0
```

Figure 11: VGG16 architecture model

Two connected layers are added to the VGG architecture to make the classification.

Performances




```
[ ]   1 model.evaluate(x_test , y_test)

313/313 [==============================] - 4s 13ms/step - loss: 0.0393 - accuracy: 0.9915
[0.039329979568719864, 0.9915000200271606]
```

Figure 12: Learning curves and evaluation of VGG16 model

Thus, the VGG model does better on the test set and beats the state-of-the-art model LeNet1998 with an accuracy of 99.15% against 98.7% for LeNet.

Upon conducting cross-validation, the results show that the model did not overfit during the 10 configured epochs. Both the training and validation accuracy curves remain close to each other, with only minimal Although the optimal number of epochs could possibly be higher than 15, the current setting already provides excellent results and would result in a significantly longer learning time. Therefore, 15 epochs have been chosen as the final value.

Conclusion

In conclusion, the fully connected, CNN, and VGG models performed very well on the MNIST dataset due to its simplicity. VGG proved to be the best model, beating the LeNet1998 model, but it may be overkill to use such a complex model for this simple handwritten digit recognition task. It is important to consider the complexity of the model relative to the complexity of the data when determining the most appropriate model for a given task.

Task 2: UAV Object Detection using transfer learning

Objectives

The objective of this task is to train an object detector that achieves good performance on the small UAV image dataset provided. To do this, we will use a transfer learning method using two different object detection networks, namely YOLOV5 and VGG. The objective is to compare the performance of the two networks in correctly classifying the position of UAVs in the images. The final goal is to choose the model that provides the best performance for UAV detection.

Approach

Two models will be trained for drone detection:
- Transfer Learning with VGG16
-YOLOV7 algorithm

After litterature review, some axes can already be drawn:

- YOLOv7 is considered a more effective option than VGG16 for drone detection for several reasons:

- YOLOv7 is an object detection model that uses a deep convolutional network (CNN) to detect objects in images. It is specifically designed for fast and accurate object detection, making it a suitable choice for drone detection.

- VGG16, on the other hand, is an image classification model that is designed to classify images into different categories. It is not designed for object detection, which can make it more difficult for drone detection.

- YOLOv7 uses algorithms such as YOLO (You Only Look Once) and anchor boxes to perform object detection, which allows it to detect objects faster and with increased accuracy compared to VGG16.

- In addition, YOLOv7 is capable of processing real-time images, making it an ideal choice for real-time applications such as drone detection.

- In summary, YOLOv7 is a more suitable option for drone detection due to its ability to detect objects quickly and accurately, as well as its optimization for real-time applications.

2.a) Sample examples

Figure 13: 3x4 subplot of random drone images with bounding box associated

After selecting 12 images at random, we can check that the bounding boxes are pointing to the drones in each image. This means that the data has been imported for the next step. It is interesting to note that not all images in the dataset are the same size and that resizing of the images and bounding boxes will be necessary.

The percentage of data used for the training set and the test set is 80% for training and 20%. Of the 20% of the validation set, it is partitioned into two to create a test set in addition to the validation set. Thus 30% of the initial validation set is used for the test phase. It is important to randomly select the data for each set to avoid bias in the evaluation of the model.

2.b) VGG16 implementation

When transfer learning with VGG16, we will use the basic layers of VGG16 that are common to all images (such as filters for shape and texture features), while adding new layers that are specific to drone recognition. We will then train this new model on a dataset including drone images so that it can learn to recognize these objects. This approach will allow us to take advantage of the power of VGG16 while adding new features for drone recognition.

In the recognition algorithm, the optimization of weights and parameters is used to minimize the loss function. For the task of object localization, which is a regression problem, a regression loss function is required. In this case, the L2 distance, also known as Mean Squared Root (MSE) distance, is utilized.

```
Model: "model_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_5 (InputLayer)         [(None, 224, 224, 3)]     0
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
flatten_4 (Flatten)          (None, 25088)             0
dense_20 (Dense)             (None, 256)               6422784
dense_21 (Dense)             (None, 128)               32896
dense_22 (Dense)             (None, 64)                8256
dense_23 (Dense)             (None, 32)                2080
dense_24 (Dense)             (None, 4)                 132
=================================================================
Total params: 21,180,836
Trainable params: 6,466,148
Non-trainable params: 14,714,688
```

Figure 14: VGG16 based model summary

We use the pre-trained weights of VGG16 (red box) to which we add a fully connected (black box) training model. The last layer has 4 neurons, because there are 4 coordinates for the bounding box.

Learning Curves

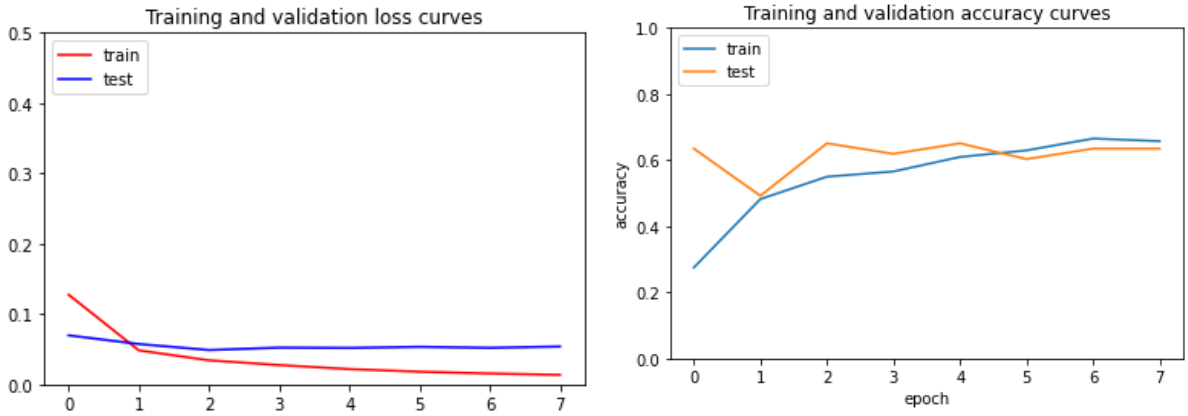


Figure 15: Learning curves of the VGG16 based model

The learning of this model seems quite satisfactory, it delivers quantitatively average performances in terms of accuracy, the curves converge.

We obtain an average accuracy of 62% at the end of the training, which seems reasonable. The most important thing is to check qualitatively if the model detects the drones with the predicted bounding boxes

Transfer learning with VGG16 may not be the best option for drone detection because VGG16 has been trained on a wide variety of images, including animals, vehicles, buildings, etc. Therefore, the features learned by VGG16 may be very general and not particularly suitable for drone recognition.

Metrics

IOU (intersection over union) is a performance measure commonly used to evaluate the performance of an object recognition method in images. It measures the similarity between the predicted bounding boxes and the actual bounding boxes in the image. This metric is determined by calculating the area of intersection between the two bounding boxes and comparing it to the area of union of the two boxes.

The advantage of this metric is that it takes into account both the accuracy of the bounding box position as well as the size of the bounding box. This means that a correctly positioned but incorrectly sized bounding box will have a lower IOU score than a correctly positioned and appropriately sized bounding box.

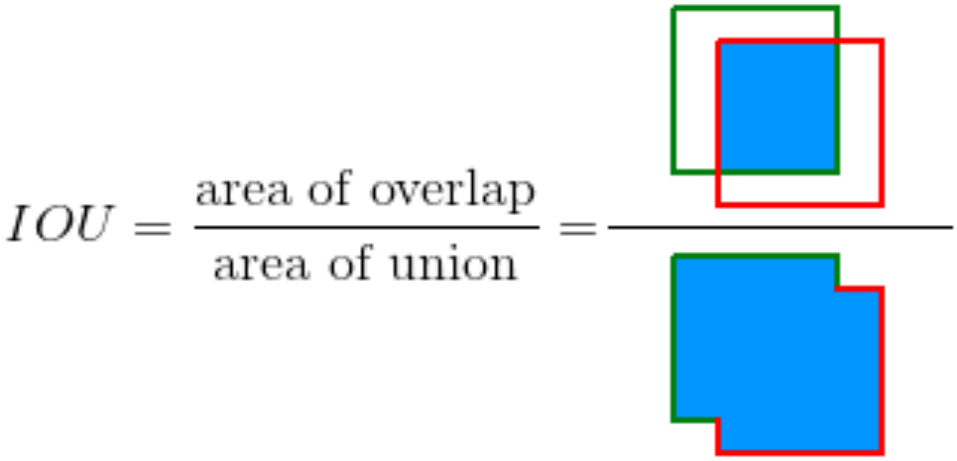


Figure 16: Illustration of the IOU metric

However, it is important to note that the IOU metric can be sensitive and it is possible to recognize an object even if the IOU value is low. For example, if the predicted bounding box is very close to the actual bounding box, but the size of the predicted bounding box is slightly different, the IOU value will still be high and the model will be considered to have correctly identified the object. Therefore, it is important to consider other factors such as the accuracy of the object class prediction when evaluating the object recognition performance.

Test on unseen drone pictures

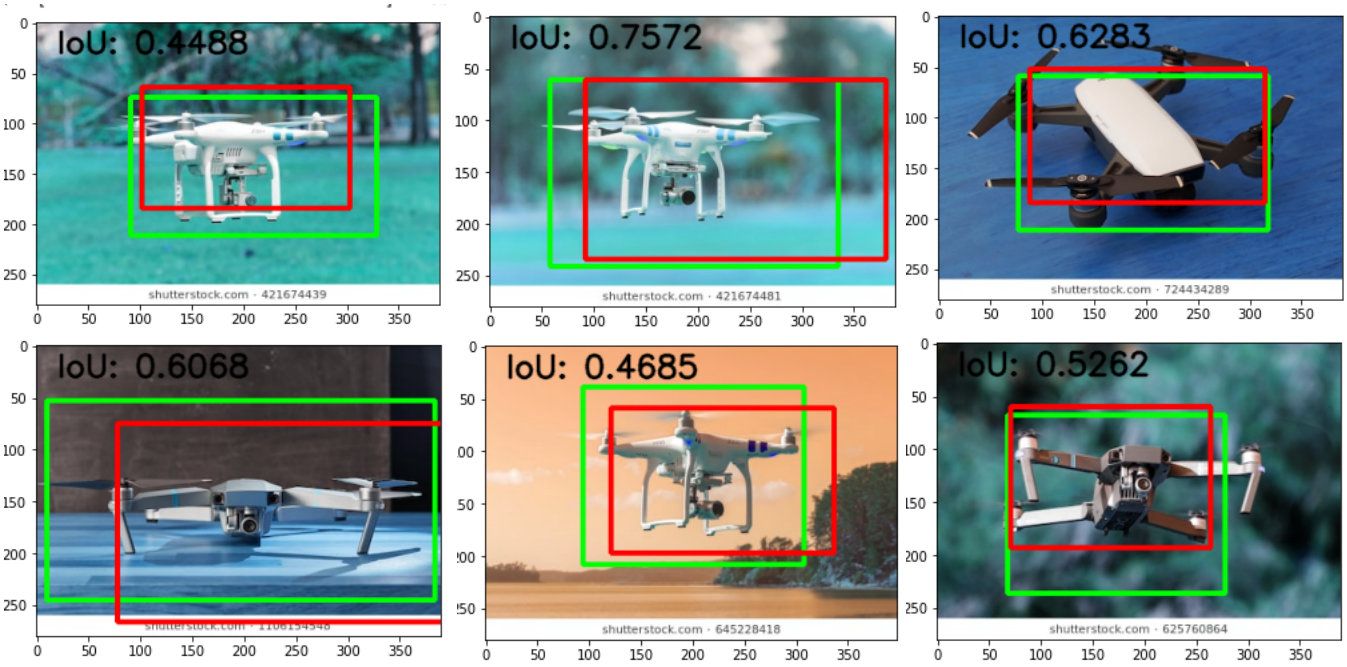


Figure 17: Test on unseen drone images of the VGG16 based model

As seen on the test images, the model is able to recognize the drones (box red) but there is often an offset which considerably lowers the IOU value as explained above. Objectively, the objective of drone detection is fulfilled with this model. It is interesting to compare it to a state-of-the-art model like YOLOV7 now.

2.c) YOLO implementation

YOLOv7 is an object detection algorithm based on deep learning. It was designed to be fast and accurate, which makes it ideal for real-time applications.

The principle of YOLOv7 is to divide an image into several sub-regions and to predict for each sub-region the classes of objects present and their positions. The algorithm uses a convolutional neural network to make these predictions and to learn the characteristics of the objects to be detected.

In the case of drone recognition, YOLOv7 can be used to detect the presence of a drone in an image and to predict its position. This can be useful for security applications, such as airspace surveillance, or for commercial applications, such as drone package delivery.

The advantage of YOLOv7 for drone recognition is its speed and accuracy. It can detect objects quickly and with very good accuracy, making it an ideal choice for real-time applications. In addition, by using deep learning, YOLOv7 can adapt to different lighting conditions and drone shapes and sizes, further enhancing its suitability for this task.

Pre-Processing

With the given dataset, the annotations are stored as (class, x, y, width, height), where x and y are the coordinates of the top left corner of the bounding box, and width and height are the width and height of the bounding box.

The normalization performed is to divide each coordinate by the width and height of the image. The purpose of this normalization is to put the coordinates back into a normalized space, where the width and height of the image are each normalized to 1.

First, the centroid of the bounding box is calculated by adding x and half the width of the bounding box, then dividing this result by the width of the image. The same process is repeated for y and the height of the bounding box.

Finally, the width and height are also normalized by dividing them by the width and height of the image, respectively. This normalization ensures that the bounding boxes are always proportional to the image, regardless of the image size.

```
classe, x, y, width, height = map(float, line.strip().split())
x = (x+width/2)/ widthy
y = (y+height/2)/heighty
width /= widthy
height /= heighty
```

Figure 18: Normalization code

To fit with YOLOv7 inputs, all images are resized to 448*448 square format.

Learning curves

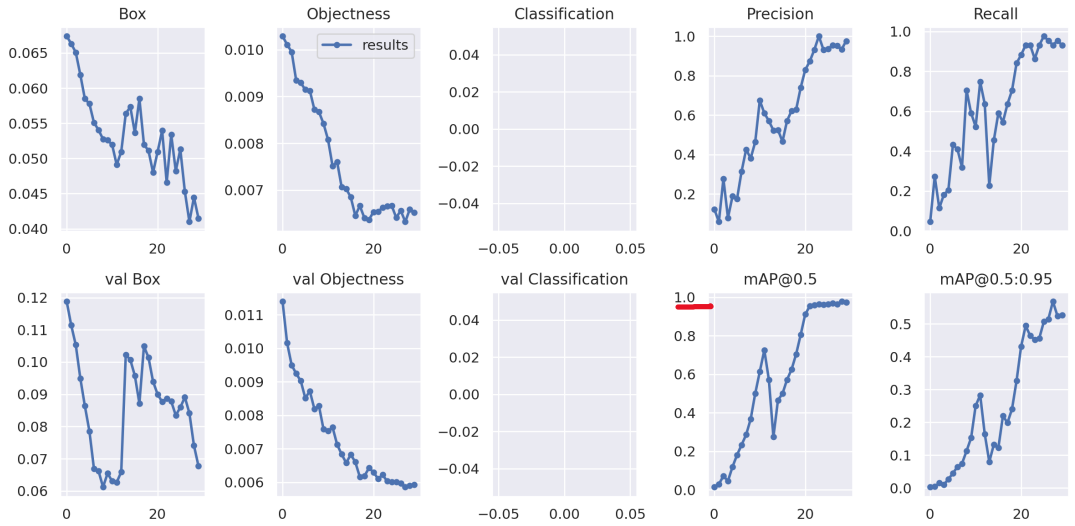


Figure 19: learning curves of YOLOV7 training

```
Epoch   gpu_mem       box       obj       cls     total    labels  img_size
28/29     10.7G   0.04447  0.006594         0   0.05106        53       448: 100% 8/8 [00:09<00:00,  1.14s/it]
           Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100% 1/1 [00:00<00:00,  1.11it/s]
             all          44          44       0.932       0.955       0.978       0.524
```

Figure 20: Best learning epoch of YOLOV7 training

The curves all converge although there are probably sometimes overfitting during the learning process (sudden unexpected peak). This model performs much better quantitatively than the previous one, which was expected from the beginning following the literature review. The performances reached are even close to perfection on all the curves. All curves converge although the curves are sometimes noisy. The most important metric that will be detailed next is the mAP@.5 . We obtain a very satisfactory performance of 97.8%, much better than the model based on VGG16.

Metrics

The mAP@.5 (mean average precision at 0.5 IoU) is a metric commonly used to evaluate the performance of an object detection model. This means that if the map@.5 is almost equal to 1, it means that the model has a very high detection accuracy for the objects of interest.

The mAP@.5 measures the average accuracy at 0.5 IoU for all categories. IoU (intersection over union) is a measure that calculates the proportion of the intersection between the predicted region and the actual region of an object compared to the united region of these two regions. If the IoU is greater than 0.5, it means that the predicted region of the object is considered to be correctly detected.The mAP@.5 metric is used to evaluate the performance of a model because it measures the detection accuracy for a given region, which is crucial for many applications such as road traffic recognition or traffic light recognition.

In the case of the drone detection learning above, a mAP@.5 almost equal to 1 means that the model has very good accuracy in detecting objects of interest.

After a quantitative evaluation of our model, it is possible to check on the test dataset whether the model detects drones.

Test on unseen drone pictures

Figure 21: Test on unseen drone images of the YOLOV7 model

Qualitatively, the results are more than satisfactory on the 14 test images. The bounding boxes are well centred on the test images. The model is ready to be deployed in a physical architecture

Conclusion

In this project, two different tasks were tackled. In the first task, we utilized a well-known classifier for recognizing handwritten digits and achieved outstanding results without the need for a deep neural network. In the second task, we had to tackle a more complex challenge, requiring the use of more advanced models and settings. To this end, we utilized transfer learning by retraining a pre-trained VGG16 model with a specific dataset to predict the location of unmanned aerial vehicles (UAVs) in pictures through regression. Moreover, we also trained a Yolov5 algorithm for the UAV recognition task to compare the performance of the two approaches.To achieve this, it was necessary to carefully manage various parameters, such as controlling the size of the images to ensure the accuracy of the predictions.Additionally, it is equally important to conduct thorough pre-processing of the data, as this can greatly impact the accuracy of the model's predictions.

Bibliography

[1] Redmon, J., & Farhadi, A. (2017). YOLO9000: Better, Faster, Stronger. arXiv preprint arXiv:1612.08242.

[2] Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

[3] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).

[4] Deng, L., Dong, W., Socher, R., Li, L. J., Li, K., & Fei-Fei, L. (2009, June). Imagenet: A large-scale hierarchical image database. In Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on (pp. 248-255). IEEE.

[5] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning (Vol. 1). Cambridge: MIT press.

[6] Shao, L., & Liu, Y. (2018). Drone detection using deep learning. arXiv preprint arXiv:1807.10721.

[7] Ouyang, W., & Wang, X. (2018). Multi-object tracking with quadrilaterals. arXiv preprint arXiv:1811.08883.

[8] Voulodimos, A., Doulamis, N., Gasteratos, A., & Kollias, S. D. (2018). A review of deep learning techniques for drone-based object detection and segmentation. arXiv preprint arXiv:1810.12824