By: Zaineh Mihyar

Developing a deep learning-based system

Technical Report

Table of Contents

[1 Problem statement 2](#_Toc200815305)

[2 Research on Neural Networks and Architectures 2](#_Toc200815306)

[2.1 Neural Networks used for the problem 2](#_Toc200815307)

[2.2 Modern architectures 2](#_Toc200815308)

[2.3 Modern architectures comparison 7](#_Toc200815309)

[3 Models’ development and training 9](#_Toc200815310)

[3.1 Dataset 9](#_Toc200815311)

[3.2 Training and validation 11](#_Toc200815312)

[4 Models’ testing and evaluation 19](#_Toc200815313)

[4.1 Testing 19](#_Toc200815314)

[4.2 Over/under-fitting assessment 23](#_Toc200815315)

[4.3 Results analysis 24](#_Toc200815316)

[4.4 Effectiveness assessment 25](#_Toc200815317)

[4.5 Interface development 26](#_Toc200815318)

[4.6 Critical evaluation of models 27](#_Toc200815319)

[References 28](#_Toc200815320)

# Problem statement

The project aims to develop deep learning models that can classify remote sensing (RS) satellite images (collecting earth images without physical contact) into four categories: cloudy, desert, green area, and water. The model should process input satellite images, identify visual patterns, and correctly assign each image to its category.

# Research on Neural Networks and Architectures

## Neural Networks used for the problem

Alkhelaiwi et al. [1] have proposed a deep learning framework with privacy preservation for satellite image classification. The authors used a custom Convolutional Neural Network (CNN) and applied pre-trained CNN architectures, including VGG16, ResNet50, Xception, and DenseNet121. The custom CNN model has 3 convolution layers (32 filters for the first 2 layers, and 64 for the last layer), 3 polling layers, a dropout layer, a flattening layer, 2 fully connected layers (64 and 4 neurons), and an activation function, ReLU and SoftMax. The model was trained on 128×128×3 images using stochastic gradient descent (SGD). Part of their approach is a homomorphic encryption scheme (Parlier encryption) integration, which allows training encryption data with no exposure of confidential information. The results showed strong classificatory outputs on a real-world satellite set from regions of Saudi Arabia.

M. Pritt and G. Chern [2] proposed a deep learning system based on an ensemble of Convolutional Neural Networks (CNNs), which input the satellite image with metadata, then classifies the images into 63 classes, including (airport, airport terminal, lighthouse, tower, and racetracks). To enhance the system’s performance, deep CNN architectures (DenseNet-161, ResNet-152, Inception-v3, and Xception) were used in the ensemble. The system achieved an accuracy of 83% and an F1 score of 0.797, with 15 of the 63 classes classified at 95% accuracy or higher.

YadavIn et al. [3] proposed a study on satellite image classification, using Convolutional Neural Networks (CNNs) to classify satellite images based on topological and geographical features into 10 classes. 3 pre-trained CNN architectures: ResNet50, ResNet101, and GoogleNet, were used in this study, to enhance each model, sequence layers were added, and the data was pre-processed using LAB channel operations. The results showed that GoogleNet achieved the best performance with an accuracy of 99.68%, a precision of 99.42%, a recall of 99.51%, and an F1-score of 99.45%. The authors observed that deeper networks like ResNet101 do not necessarily perform better than shallower models, and that GoogleNet's 22-layer CNN is more efficient for satellite image classification.

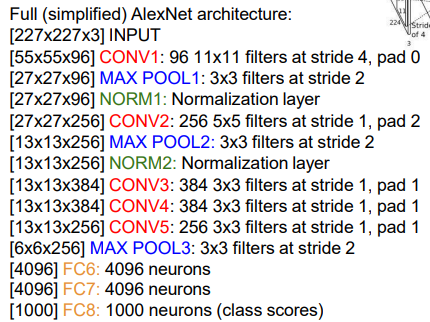
Adegun et al. [4] evaluated deep learning models for classifying high-resolution satellite images. They used Convolutional Neural Networks (CNNs) and Vision Transformers, assessing their performance on 3 datasets. CNN architectures used are (ResNet, DenseNet, EfficientNet, VGG16, and InceptionV3). The results showed that DenseNet121, ResNet101, and Vision Transformer achieved the best performance, with accuracies, precisions, F1-Scores, and recalls exceeding 90%, and stable losses during training.

## Modern architectures

After some research, Convolutional Neural Network (CNN) is the best neural network used for classifying remote sensing (RS) satellite images. Below, I will be explaining some CNN architectures.

**AlexNet**

AlexNet was one of the first CNN architectures was founded by Alex Krizhevsky, and won first place in ILSVRC’11, achieving a 16.4% top 5 error, as it significantly outperformed the 2nd place achieved only 25.8%. AlexNet has only 8 layers (5 convolutional layers, 3 fully connected layers), and it uses 2 normalization layers (which aren’t used anymore). The image input size is 227x227x3. It uses different filter sizes and strides. AlexNet is not used nowadays [5].



**ZFNet**

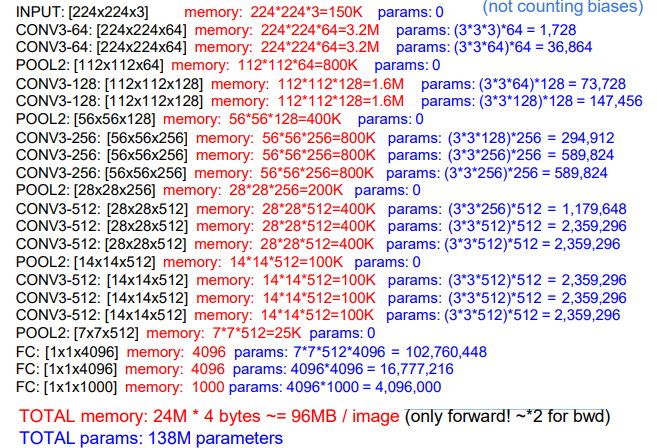
ZFNet has the same architecture as AlexNet, but it changed the filter size and stride in convolutional layers 1,3,4,5. Convolutional layer 1 changed from (11x11 stride 4) to (7x7 stride 2). Convolutional layer 3,4,5 instead of 384, 384, 256 filters used 512, 1024, 512. These changes made it win ILSVRC’13 with a top 5 error of 11.7% [5].

**VGG16**

Visual Geometry Group (VGG) is a deep convolutional neural network architecture that was introduced by the University of Oxford in 2014, it contains 13 convolutional layers and 3 fully connected layers, making it deeper than other models available at the time. VGG16 was developed to simplify the design of CNNs by using 3x3 filters throughout the architecture, so it can detect features more effectively than using larger filters with fewer layers. Using this approach reduces the number of parameters (138 million parameters) and enables deeper networks to be trained effectively. VGG16 processes input images through a series of convolutional layers followed by max-pooling layers, gradually reducing the dimensions while increasing depth, and ends with fully connected layers for classification. It is widely used in research and industry for feature extraction tasks, including satellite image classification [6].

A group of yellow and white rectangular objects

AI-generated content may be incorrect.



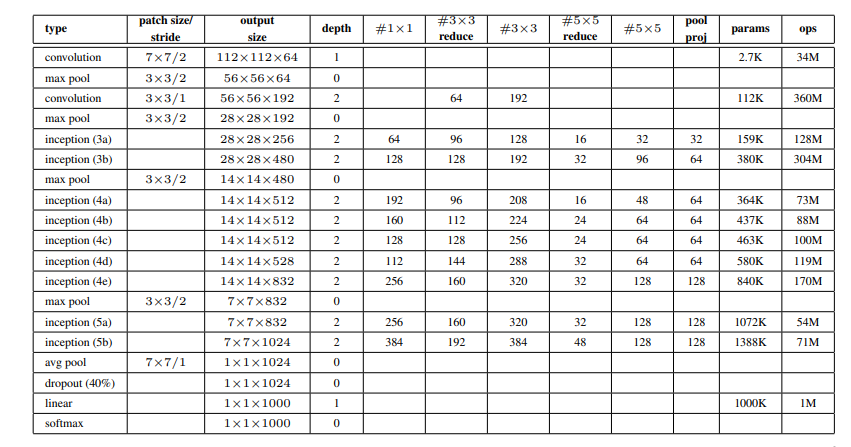
**GoogLeNet**

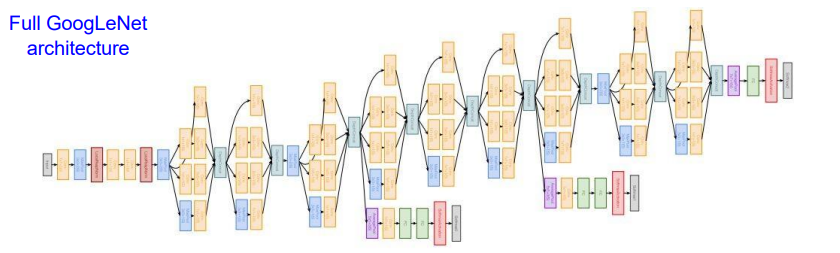
GoogLeNet is a deep architecture introduced in 2014 in the research paper “Going Deeper with Convolutions”, developed by researchers at Google and various universities, to also solve the problem of vanishing/exploding gradients. It won ILSVRC’14 6.7% top 5 error. Its architecture is made up of 22 layers, including 9 Inception modules, that may be separated by pooling layers, and it has no fully connected layers [7]. This architecture reduces the number of parameters while extracting deep features and maintaining good computational efficiency. It is suitable for satellite image classification, where different land types can exhibit patterns of various shapes and sizes [8].

A diagram of a diagram

AI-generated content may be incorrect.

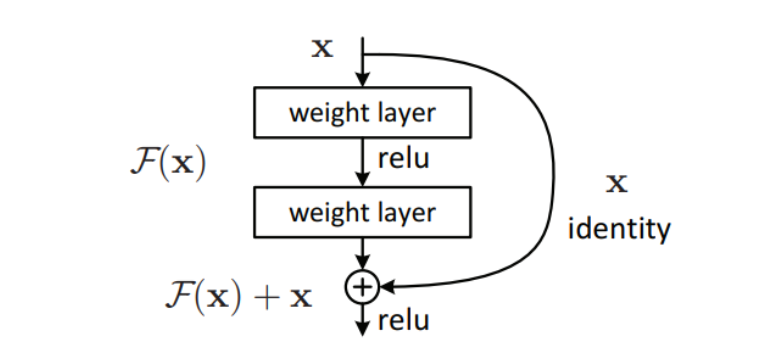
The diagram illustrates an Inception module, which processes the output from the previous layer through 4 parallel paths to extract features. The first path uses a 1×1 convolution directly to learn small details and reduce dimensionality. The other 2 paths apply a 1×1 convolution one followed by a 3×3 and the other by a 5×5 convolution, to extract medium and large patterns, while keeping computational costs low. The last path applies 3×3 max pooling to summarize nearby pixels by picking the max value, followed by a 1×1 convolution to transform the output to maintain uniform depth and learn something new. The outputs from all these paths are then concatenated, which enables the network to analyse information from multiple paths, enhancing performance while reducing the number of parameters compared to traditional architectures [9].





**ResNet**

Residual Network is a type of deep learning model that was introduced in 2015 to fix the vanishing/exploding gradients problem, where the gradient either decreases (the model stops learning) or increases (makes learning unstable) the gradient dramatically. The problem was solved by introducing the concept of residual blocks, which added a skip connection, meaning if a few layers aren’t helpful, the network skips them and connects the output directly to a deeper layer. How does that help? It just learns what needs to be changed, not learn from scratch, “learns small residuals”. ResNets made training very deep networks easier by maintaining stable gradients during backpropagation, preventing them from vanishing or exploding. This architecture facilitates powerful learning in complicated image sets, such as satellite images. The variants like ResNet-50 and ResNet-101, which balance depth and cost-efficiency, make them popular. ResNet won ILSVRC’15 with only 3.57% top 5 error [10]. The



ResNet architecture has a stack of residual blocks, each block has 2 (3x3 convolutional layers), no fully connected layers at the end only 1 to output classes.

**DenseNet**

The Densely Connected Convolutional Networks were introduced by Huangin et al. in their paper “Densely Connected Convolutional Networks” in 2017. What is unique to DenseNet is the “dense connectivity”, which links each layer to each other by passing the output to all the subsequent layers, not just to the next, thereby enabling each layer to obtain feature maps from all the preceding layers as input. In this way, the number of parameters is decreased while the gradient flow is improved [11].

DenseNet uses multiple dense blocks separated by transition layers, which help control model complexity through 1×1 convolutions and average pooling. A key hyperparameter in DenseNet is the growth rate, which determines how many new features maps each layer adds. DenseNet-121 is the most used version, which consists of four dense blocks with 121 layers in total. DenseNet is suitable for satellite image classification as it can capture the textures and patterns, which are ideal for distinguishing similar land images, such as desert and cloud regions [12].

**Network in Network (NiN)**

NiN is a method founded by Lin, Chen, and Yan in 2013 to strengthen the abilities of traditional CNNs. Rather than using standard linear filters inside standard CNNs, NiN uses MLPs within local areas instead. Using 1×1 convolutions serve the purpose of letting MLPs work independently on the channel dimension, making it possible for features to be transformed in complicated and nonlinear ways in every receptive field. The main goal of NiN is to boost how well a model can model data, without adding significant complexity or expenses. NiN also applies global average pooling at the end instead of a fully connected layer, allowing there to be fewer layers and reducing the chances of overfitting while also improving invariance to translation. NiN has led to better results in image classification than regular CNNs, and at the same time requires less memory. It set the stage for architectures such as GoogLeNet and SqueezeNet by encouraging the use of 1×1 convolutions and global average pooling as good options instead of huge fully connected layers [13].

## Modern architectures comparison

Table : Modern architectures used to solve the problem.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Architecture** | **Description and number/types of layers** | **Advantages** | **Disadvantages** | **Justification** |
| DenseNet | Uses dense connections; each layer receives input from all previous layers. Popular versions: DenseNet-121, DenseNet-169. | - Efficient gradient flow - Reduces overfitting - Fewer parameters due to feature reuse | - Memory intensive - Complex implementation due to dense connections | Useful for remote sensing tasks with small to medium datasets, as feature reuse helps preserve spatial detail and reduces overfitting. |
| ResNet | Introduced Residual Blocks with skip connections. Popular versions: ResNet-50, ResNet-101, ResNet-152. | - Solves vanishing gradient - Enables very deep networks - Stable training | - Computationally expensive - May overfit on small datasets | Suitable for very deep models where high feature extraction is needed; stable during training and accurate on complex patterns. |
| VGG16 | 13 convolutional layers + 3 fully connected layers; all conv layers use 3×3 filters | -Simple and consistent architecture -High accuracy - Good for transfer learning | - Large number of parameters  -Requires high computational resources | Good baseline model; simple to implement and effective for transfer learning in remote sensing classification tasks. |
| GoogleNet | 22 layers deep with Inception modules using parallel 1×1, 3×3, and 5×5 convolutions | - Fewer parameters than VGG - Captures multi-scale features - Efficient for deep learning | - Complex architecture - Inception modules can be hard to modify | Useful for capturing features at multiple scales, which is essential for diverse terrain patterns in satellite images. |
| AlexNet | 5 convolutional layers + 3 fully connected layers; ReLU activation, dropout, overlapping max pooling | - Introduced deep CNN to large-scale tasks  -Faster training with ReLU  - Uses data augmentation and dropout | - High memory usage  - Shallow compared to modern networks  - Fixed input size |  |
| ZFNet | Extension of AlexNet with 5 conv + 3 FC layers; optimized filter sizes and strides based on visualization | - Better accuracy than AlexNet  - Visualizes feature maps for better understanding  - Reduced overfitting via tuning | - Still shallow  - Slightly higher complexity than AlexNet |  |
| NiN | MLPConv: 1×1 convolutions after standard convolutions; ends with global average pooling instead of FC layers | - Parameter efficient  - Enhances local feature abstraction  - Reduces overfitting via global average pooling | - Less powerful on very large datasets  - Less popular than other modern architectures |  |

[14] ‌

After reading in the literature, the most used architectures for such a problem are DenseNet, ResNet, VGG16, and GoogLeNet.

DenseNet performs well in remote sensing, since training data is usually limited in this field. Each of its layers has every previous layer’s output as its input. Such a structure leads to the maximum use of common features and helps learning them well from small and medium-sized datasets. Besides, DenseNet can be made with fewer parameters as it skips unnecessary learning of similar features. Since detecting minor differences in texture or colour can be vital for satellite images, DenseNet can maintain spatial details through its network, improving the system’s ability to locate and identify objects in images. Because feature concatenation uses more memory, the use becomes worthwhile because of the advantages of better-represented and less complex models.

Many deep networks use ResNet, a highly reliable deep architecture. The residual block is its main invention, since it connects parts of the model so that features and gradients can be passed straight through several layers. Because of residual blocks, the issue of vanishing gradients no longer holds back networks from having many layers. ResNet is designed to identify important information from big parts of an image as well as small sections. ResNet can be used in pretraining and training for multiple tasks and datasets, which is why it is considered practical. Because it is expensive and complicated to operate, this kind of network should be used for tasks that call for precise and dependable results.

GoogLeNet, developed on the Inception framework, makes it easy to get a lot of information from images. Every Inception module does a set of convolutions all at once (1×1, 3×3, 5×5), so the image content gets analysed on various spatial scales. Besides, GoogLeNet depends less on training parameters than VGG16, as 1×1 convolutions help to reduce the data’s dimensions. It makes the model more suitable for being used on devices with not much processing power. Even though it is hard to adapt, the architecture’s great advantage is that it works excellently with complex and diverse image content.

VGG16 is used for being straightforward and has the same structure throughout. The model is designed with 13 convolutional layers with filters of size 3×3, followed by 3 fully connected layers. VGG16 is hard to train since it needs a lot of resources and has many parameters; still, its regular design makes it simple to set up and study, making it valuable for academic and test environments. Trained VGG16 models are usually improved by fine-tuning on smaller datasets specific to a domain and work very well. The transferability helps a lot when there is not much labelled data available. Even though VGG16 does not achieve the best speed, it is still a key part of experiments and useful with ensemble models because it delivers reliable results.

# Models’ development and training

## Dataset

I got the dataset “Satellite Image Classification (RSI-CB256)” from Kaggle, by the author is Mahmoud Reda. The dataset belongs to the remote sensing domain and is designed for land cover classification using satellite imagery. The images were collected from sensors and Google Maps snapshots. The dataset contains 5631 JPEG images classified into 4 classes (1500 cloudy images, 1311 desert images, 1500 green area images, and 1500 water images). The image resolution is 256x256.

A black background with white text

AI-generated content may be incorrect.

<https://www.kaggle.com/datasets/mahmoudreda55/satellite-image-classification/data?select=data>

**Preparing the data**

First, I resized the images to 224x224, which meets the standard size for the models used.

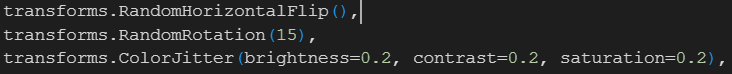


Then, I normalized the images using the mean and standard deviation, because pretrained models were trained on a specific dataset (ImageNet) using the following mean and standard deviation.



The original dataset consists of 5,631 images. When I trained the original dataset on the deep learning models, the validation and testing accuracies reached 1 (100%). Which indicates overfitting (the model memorized the data), the models were too complex with millions of parameters.

To reduce overfitting, data augmentation was applied to the training set. Each image in the training set was augmented four times; each image may have undergone a random horizontal flip, random rotation (up to 15°), and random brightness, contrast, and saturation adjustments. This resulted in an augmented training dataset 4× larger than the original training set.



A black screen with white text

AI-generated content may be incorrect.

VGG16 results before data augmentation:

A grey background with white text

AI-generated content may be incorrect.  
A screen shot of a black screen

AI-generated content may be incorrect.

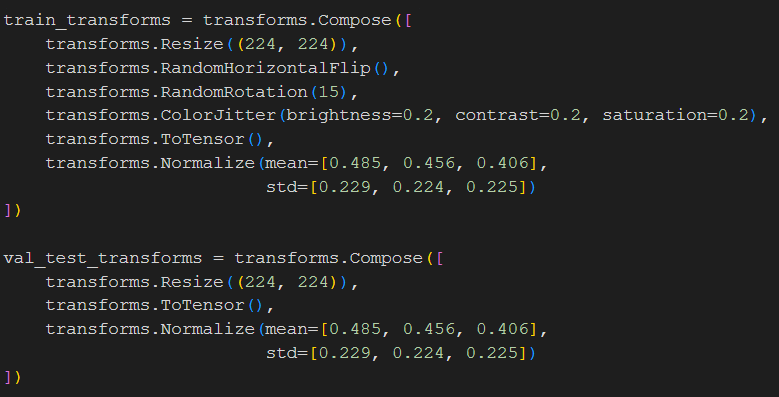
VGG16 results after data augmentation:

A screen shot of a computer

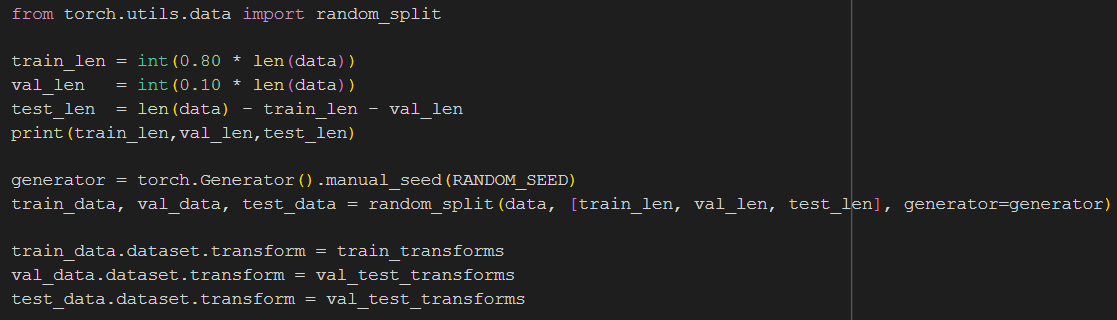
AI-generated content may be incorrect.

Better generalization

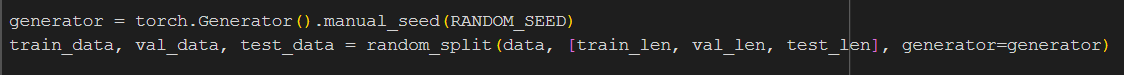
## Training and validation



Transforms.compose() combines a list of image transformations into a sequence, so they are applied in order. Each image will go through all the listed steps, one after the other.



First, I split the data into testing (10%), validation (10%), and training (80%).



Each image gets assigned once to the train, test, or validation set. Using a fixed random seed ensures the same images go into the train, validation, or test splits every time I run the code. So the results are consistent.

A black background with white text

AI-generated content may be incorrect.

Apply training augmentation to the training set only, to generalize better, but it's not used on the testing and validation, because they should evaluate the model’s performance, so we shouldn’t change it and keep it consistent.

A computer screen shot of a program code

AI-generated content may be incorrect.

The augmentation factor determines how many times I want to increase my training data. For example, if I originally have 4504 images, after augmentation, I will have 18,016 images. Augmented images will store the augmented images.

The for loop iterates 4 times (augmentation factor) random transformation is applied to each image in the training set. Deep copy ensures each dataset applies different random transformations.

A black screen with white text

AI-generated content may be incorrect.

Data loader feeds the data to the models. Shuffle = True is only used for training to avoid learning the data order.

I used 4 models VGG16, ResNet18, DenseNet121, and GoogLeNet. Each model was trained on 3 different hyperparameters.

Table : Description of the hyperparameters considered for each architecture

|  |  |  |  |
| --- | --- | --- | --- |
| **Architecture** | **Hyper-parameter** | **Description** | **Value(s)** |
| **VGG16** | Dropout | Dropout is a regularization technique used to prevent overfitting in neural networks. | 0.4,0.5,0.6 |
| Learning rate | The learning rate controls how much the model weights are updated during training. | 0.0005 |
| Weight Decay | Weight decay penalizes large weights in the model to reduce overfitting. | 0, 0.00001 |
| **DenseNet121** | Dropout |  | 0.5 |
| Learning rate |  | 0.0005, 0.001 |
| Weight Decay |  | 0.01, 0.001 |
| **ResNet18** | Dropout |  | 0, 0.5 |
| Learning rate |  | 0.0005, 0.001 |
| Weight Decay |  | 0 |
| **GoogLeNet** | Dropout |  | 0.5 |
| Learning rate |  | 0.0005, 0.001, 0.0007 |
| Weight Decay |  | 0.001, 0.00001 |

**VGG16 Training**

A computer screen shot of a program

AI-generated content may be incorrect.

The first model I used is the VGG16 pretrained model. I made some modifications, adding a dropout layer to prevent overfitting. I replaced the final layer to match the number of classes in the dataset. Then I defined the multi-class classification task loss function (crossEntropyLoss), which tells the difference between the predicted value from the actual. Then I defined an optimizer (Adam) to adjust weights to minimize loss.

A black background with white text

AI-generated content may be incorrect.

Stores the training and validation accuracies after each epoch.

A screen shot of a computer program

AI-generated content may be incorrect.

This is the training phase, set the model to training mode, initialize the loss, and create lists of predicted values and actual values.

The for loop iterates over batches of input images and their labels from the training set using the train\_loader.

optimizer.zero\_grad(), clears previously accumulated gradients from the last backward pass; if this is not done, gradients from previous steps would accumulate and corrupt the update.

outputs = model(inputs), performs a forward pass using the input images to obtain predictions.

loss = criterion(outputs, labels), compute the loss value.

loss.backward(), performs backpropagation (computes the gradients of the loss for all the model’s parameters)

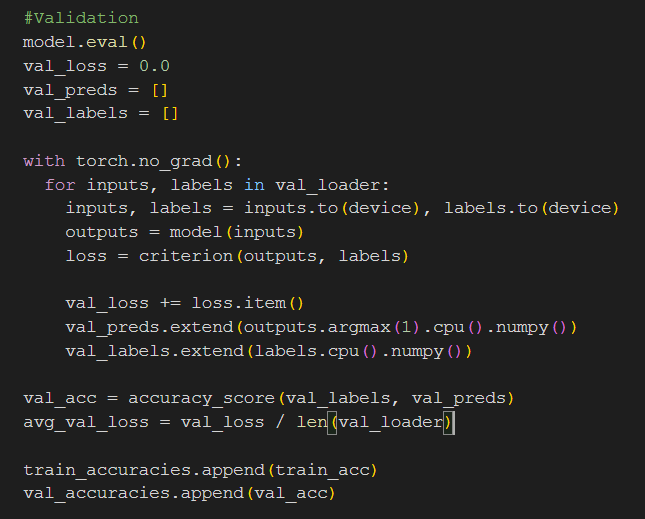
optimizer.step(), use the calculated gradients to update the model parameters (weights)

train\_loss += loss.item(), accumulate the total training loss over all batches, to calculate the average loss at the end of the epoch.

train\_preds.extend(outputs.argmax(1).cpu().numpy()), get the predicted class for each image by taking the index of the maximum logit. argmax(1) selects the class with the highest score across the class dimension.

actual\_labels.extend(labels.cpu().numpy()), convert the actual labels to NumPy format, so they can be compared with the predicted values.

Final training accuracy and loss are calculated.



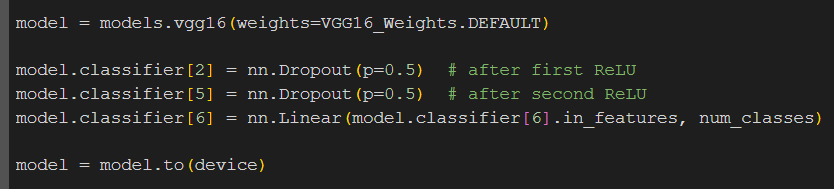
This is the validation phase, same steps as the training, but no updating weights, it just calculates the loss between actual and predicted labels.

A screen shot of a computer

AI-generated content may be incorrect.

And this is the final output of the training/validation process.

The same steps are made across all models, just the model initialization differs. In all models, I loaded the pretrained model, modified the final classification layer to fit the number of classes.



A screen shot of a computer

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

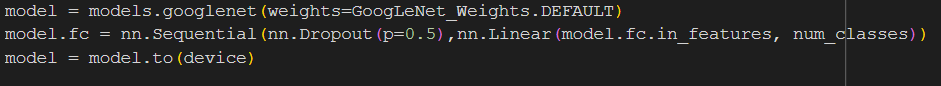


Table : Combinations of the hyperparameter values and corresponding performance achieved





**VGG16 Learning Curve**

A graph with blue and orange lines

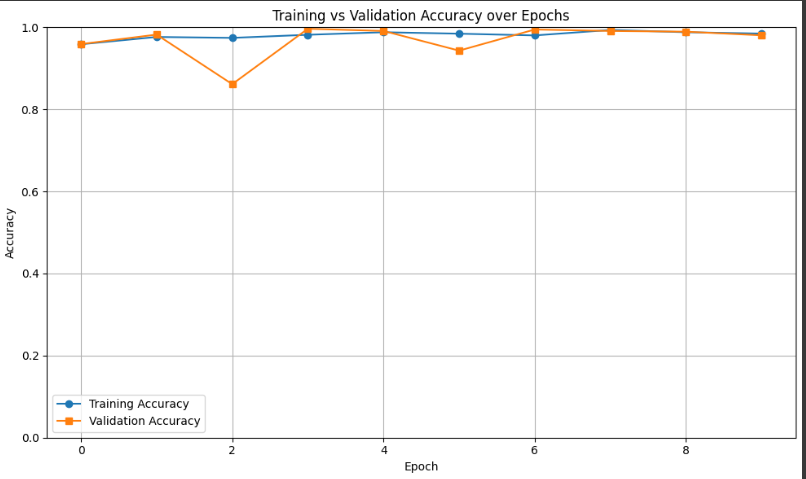
AI-generated content may be incorrect.

**ResNet18 Learning Curve**

A graph with lines and text

AI-generated content may be incorrect.

**DenseNet121 Learning Curve**

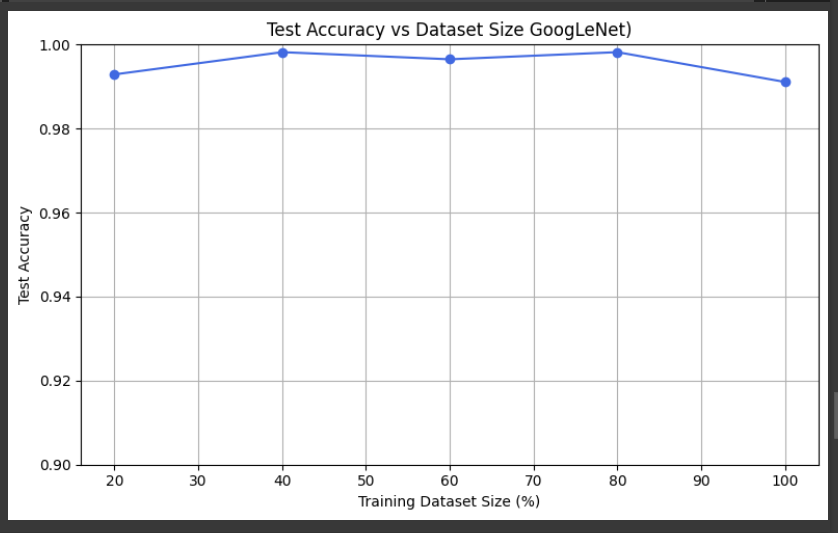


**GoogLeNet Learning Curve**

A graph with orange and blue lines

AI-generated content may be incorrect.

**Best Model Training/Validation accuracy vs dataset size**

****

# Models’ testing and evaluation

## Testing

A screen shot of a computer program

AI-generated content may be incorrect.

This is the testing phase, the first line sets the model to evaluation mode (ensures dropout is off). Then I created 2 empty lists to store model predictions and the actual labels.

with torch.no\_grad():, Temporarily disables gradient calculations, to save memory and speed up computation during evaluation (since no backpropagation is needed).

for inputs, labels in test\_loader:, loops through the test dataset in batches, inputs is a batch of test images, labels are their true values

outputs = model(inputs), runs the model on the input batch and gets raw output scores (logits) for each class.

\_, preds = torch.max(outputs, 1), for each sample, it selects the class with the highest logit value

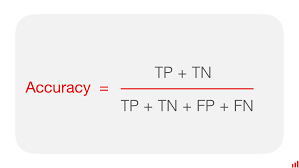
all\_preds and all\_labels are converted to NumPy format.

Then, accuracy, recall, precision, and F1-score are calculated.

**Evaluation Metrics**

Accuracy

The proportion of correctly classified instances out of the total number of instances. I used it because it’s a straightforward and commonly used metric to evaluate the overall performance of a classification model [15].



Precision

The proportion of true positive predictions out of all positive predictions made by the model. I used it because it's important in scenarios where false positives are costly or undesirable [15].

A math equation with black text

Description automatically generated

Recall

The proportion of true positive predictions out of all actual positive instances in the dataset. I used it because it’s critical in scenarios where missing positive cases (false negatives) are more problematic. For example, failing to detect a green area or water region could be more critical than mistakenly identifying one [16].

A math equation with a blue arrow

Description automatically generated

F1 Score

A means of precision and recall, balancing both metrics. I used it because it provides a single metric that considers both false positives and false negatives, and it's reliable when the dataset is slightly imbalanced, and it’s a trade-off between catching all relevant cases and avoiding false predictions [17].

A black text on a white background

Description automatically generated

Table : Values for the evaluation metrics achieved on the test set for the best model from each architecture.

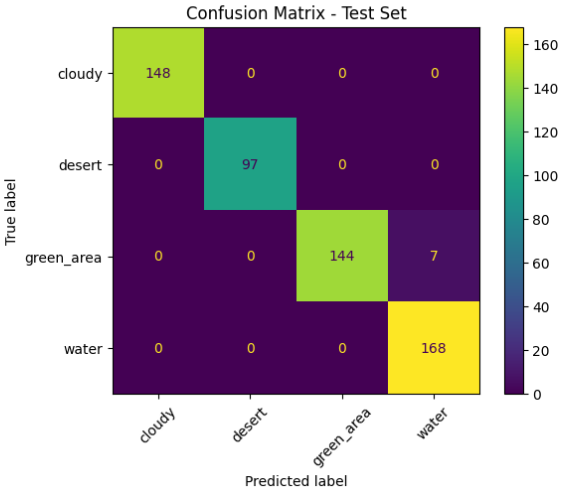


**VGG16 Confusion Matrix**

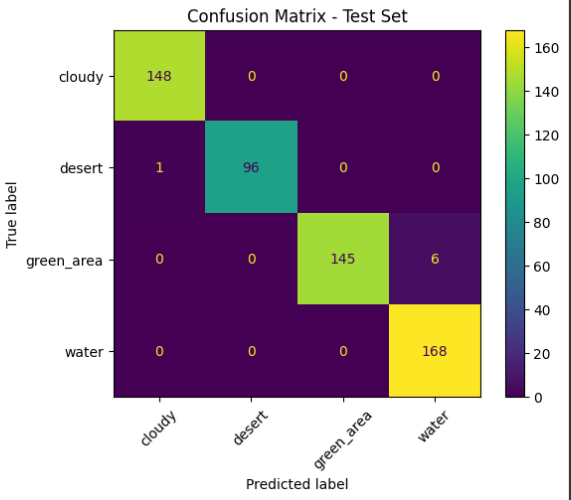
A diagram of a test set

AI-generated content may be incorrect.

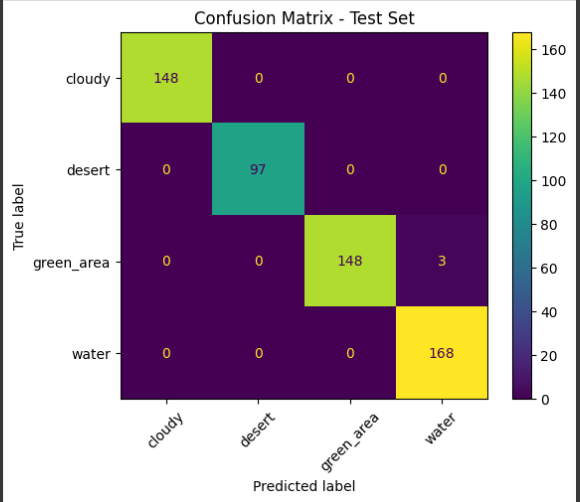
**ResNet18 Confusion Matrix**



**DenseNet121 Confusion Matrix**



**GoogLeNet Confusion Matrix**



## Over/under-fitting assessment

All 4 models show that they are balanced through training, testing, and validation, and no indication of underfitting or overfitting. Regularization methods (dropout and weight decay) were applied to all models, as well as data augmentation was used in training. These methods made the training stage more complex and enhanced generalization because they helped avoid the models being overly dependent on specific features.

**VGG16 Model**



The training accuracy is 94.26%, and validation (95%) and testing (98.94%) accuracies are slightly higher. This happens because the dropout, during training, randomly deactivates fractions of neurons (the model doesn’t rely on specific features). However, it's turned off (no neurons are dropped) during testing and validation, so the model uses the whole network to test the images. Moreover, the testing and validation images are simpler because they are not augmented, where augmented images introduce variability, making training more challenging. So, the testing and validation results indicate generalization. The precision, recall, and F1 score show that the model identifies all classes correctly, with low false positives and negatives. Overall, performance is balanced, indicating no underfitting or overfitting.

**DenseNet121**



The training accuracy (0.981) is slightly higher than the validation accuracy (0.9689); however, this doesn’t indicate overfitting, as the difference is almost 0.0121. The model is learning well and generalizing effectively. The high F1 (0.9884) and balanced metrics confirm no underfitting. All evaluation metrics are very close to each other; the model is not biased toward any specific class.

**ResNet**



The training accuracy (0.9888) is slightly higher than the validation accuracy (0.9813), this small difference (0.0075) doesn’t indicate overfitting. Precision, recall, and F1 are high and consistent, indicating robust prediction capability across all classes. Overall, very strong generalization and balanced performance, and there are no signs of underfitting.

**GoogLeNet**



The training (0.9859) and validation (0.973) accuracies are the highest among all models. As well as the testing accuracy is (0.9982). Dropout helped reduce overfitting during training, while being deactivated during validation and testing, contributing to the test results. The test accuracy is much higher than both training and validation, which can indicate that the model generalizes well. However, this indicates that the test set is easier or less diverse than the training (trained data is augmented, making it more complex).

## Results analysis

The four models' testing results demonstrate that all architectures achieved high performance across accuracy, precision, recall, and F1 score, which measures the ability to generalize to unseen data.

The overall testing performance was the best in GoogLeNet, with the test accuracy of 99.82%, precision of 99.85%, recall of 99.83%, and F1 score of 99.84. These measures show that GoogLeNet not only predicts well but also ensures that false positives and false negatives are balanced, and thus, it is the most stable model about the quality of final prediction.

The VGG16 model also showed high results, achieving a test accuracy of 98.94% and an F1 score of 99.04%. The recall was highest (98.99%) for the models, indicating that it is particularly effective in capturing most of the true positive cases.

DenseNet121 gave reliable and high-quality results, accuracy (98.76%), precision (98.97%), recall (98.75%), and F1 score (98.84%). The balanced value of precision and recall shows that DenseNet performs well in predicting false positives and false negatives.

ResNet matched DenseNet’s test accuracy (98.76%) and slightly surpassed it in F1 score (98.9%), suggesting slightly better balance in overall classification performance. Its recall (98.84%) and precision (99.0%) also indicate a high capability of finding true positives and minimizing false positives.

**Describe how, based on the performance measures, you were able to enhance the models.**

The models gave these results after a lot of trials. First, the dataset was too small, with almost 6K images, which showed (training, validation, and testing accuracies = 1) the models were very large, and the dataset was very small, so I did data augmentation, increasing the images into almost 18K so the results became more reliable. Moreover, I had tuned dropout, weight decay, and learning rate to get perfect results.

Some of the trials, when I tried 0.4 dropout and 0.001 learning rate in the VGG16 model, the results were so bad, the testing evaluations were around 0.3. After many trials, I found that the perfect learning rate for all models was 0.0005, making the model stable.

## Effectiveness assessment



As a measure of the effectiveness of the architectures, I compared them in terms of memory usage (number of parameters and model file size) and computational requirements (training time in seconds).

**VGG16**

VGG16 was the most memory- and computation-intensive model. It contained 138,357,544 parameters and a model file size of 524,531 KB. It also took the longest training time of 3630.59 seconds. While VGG16 has good accuracy, its large number of parameters and long training time make it unsuitable for real-time applications. Its old architecture does not optimize memory or computational efficiency.

**DenseNet121**

DenseNet121 showed better efficiency, with 7,978,856 parameters, a model size of 27,780 KB, and a training time of 560.41 seconds. Although having fewer parameters and model size, DenseNet still had good results, proving its efficient architecture. DenseNet reuses features, which makes the number of parameters smaller and the generalization ability good, although the computational cost is still moderate because of the complicated connectivity.

**ResNet**

ResNet offered the most balanced performance overall. It had 11,689,512 parameters and a model size of 43,745 KB, with the shortest training time of only 241.64 seconds. Its residual connections make training of deep networks easier and faster to converge. This makes ResNet very useful in situations where speed and accuracy are both needed, and on systems with limited memory and computing capability.

**GoogLeNet**

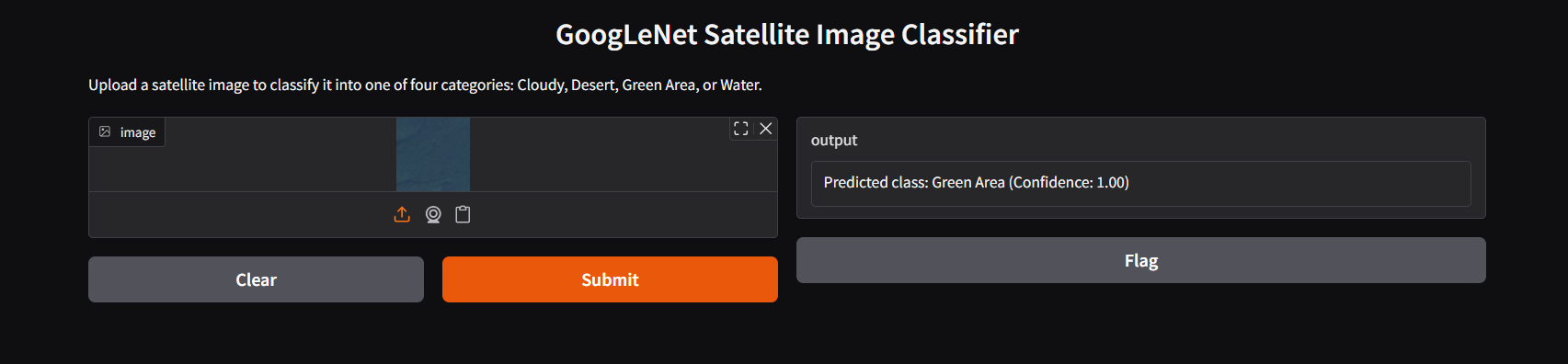
GoogLeNet was the most memory-efficient model, with 6,624,904 parameters and a model file size of only 22,071 KB. Nevertheless, its training time took 1399.72 seconds, much higher than ResNet and DenseNet. This requires more training time because its inception modules are more complex, and thus, despite improving accuracy, add more computational overhead.

Based on the above, GoogLeNet is the best model with the least number of parameters and model file size; however, it has a longer training time than ResNet and DenseNet, but 1399.72 seconds is still relatively fast. Also, it has the best validation and testing accuracy.

## Interface development

The best model among the 4 models is the GoogLeNet model. It had the highest validation and testing scores. So, I downloaded and wrapped the GoogLeNet model in an interface. To make the model predict classes.

Below shows examples of how the model predicts the 4 classes correctly.



A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

## Critical evaluation of models

GoogLeNet got a consistent test accuracy with high F1 scores, meaning highly confident predictions. It was data-efficient since its performance was still high when trained on partial datasets (20%-80%). This makes it appropriate to use where a high accuracy is required, and where labelled data is limited.

A lightweight architecture is one of the strengths of GoogLeNet. Only 6,624,904 parameters and a 22,071 KB model. This enables it to be deployed on mobile, IoT, or industrial devices that do not have huge GPU resources.

Although slower than ResNet, GoogLeNet is much more efficient than traditional models, such as VGG16. The speed of inference makes it suitable for real-time or near-real-time applications. GoogLeNet is good for users who need responsive interfaces.

The model had good generalization across training fractions, and inconsistency at some fractions, which indicated that variance might be injected by randomness in sampling. This affects Scalability in dynamical datasets and can be enhanced through better sampling.

GoogLeNet is not interpretable. End-users need explainable AI qualities like heatmaps or activated areas of an image to get an idea of why a specific class was predicted. At the given moment, the model is working as a black box, especially for non-technical users.

The Gradio wrapper makes the model easy to use for non-technical users by providing a simple, interactive front end.

**Limitations**

* GoogLeNet long training durations, 1399.72 seconds (23 minutes) it is not real-time.
* The models were trained and evaluated on a single dataset; this limits the generalizability of the results.
* Explore modern lightweight and transformer-based models.
* Lack of explainability features (tools or techniques) that help users understand how the model makes its predictions. Deep learning models are "black boxes" because they perform complex feature extraction and decision-making steps that are not understandable by humans. Visualization or explain what the model is focusing on, makes users more confident in the predictions, regardless of how accurate the model is.

**Future improvements**

* Evaluating the models on multiple datasets with different characteristics (size, noise) would help assess generalization and robustness.
* Using automated tools could help find optimal combinations of dropout, learning rate, and weight decay across different architectures more efficient than manual combinations.
* Apply techniques such as model pruning, or quantization to reduce model size and speed up inference without significant accuracy loss.
* Include additional architectures, such as MobileNet, EfficientNet, SqueezeNet, and Vision Transformers (ViT). These models offer better trade-offs between performance, model size, and speed, and are especially optimized for deployment on mobile devices.
* Use visual explanation tools to help users understand model predictions and improve trust.
* Deploy the best-performing model into a live interface to assess actual runtime performance and usability.

# References

[1] Alkhelaiwi, M., Boulila, W., Ahmad, J., Koubaa, A. and Driss, M. (2021). An Efficient Approach Based on Privacy-Preserving Deep Learning for Satellite Image Classification. Remote Sensing, 13(11), p.2221. doi:https://doi.org/10.3390/rs13112221.

[2] Pritt, M. and Chern, G. (n.d.). Satellite Image Classification with Deep Learning. [online] Available at: https://arxiv.org/pdf/2010.06497 [Accessed 27 May 2025].

[3] Yadav, D., Kapoor, K., Arun Kumar Yadav, Kumar, M., Jain, A. and Morato, J. (2024). Satellite image classification using deep learning approach. Earth science informatics. doi:https://doi.org/10.1007/s12145-024-01301-x.

[4] Adekanmi Adeyinka Adegun, Serestina Viriri and Jules-Raymond Tapamo (2023). Review of deep learning methods for remote sensing satellite images classification: experimental survey and comparative analysis. 10(1). doi:https://doi.org/10.1186/s40537-023-00772-x.

[5] Deep Learning LO10 slides

[6] GeeksforGeeks (2020). VGG-16 | CNN model. [online] GeeksforGeeks. Available at: <https://www.geeksforgeeks.org/vgg-16-cnn-model/>.

[7] Educative. (n.d.). What is GoogLeNet? [online] Available at: <https://www.educative.io/answers/what-is-googlenet>.

[8] GeeksforGeeks. (2020). Understanding GoogLeNet Model - CNN Architecture. [online] Available at: https://www.geeksforgeeks.org/understanding-googlenet-model-cnn-architecture/.

[9] Bangar, S. (2024). GoogLeNet: A Deep Dive into Google’s Neural Network Technology. [online] Medium. Available at: <https://medium.com/@siddheshb008/googlenet-a-deep-dive-into-googles-neural-network-technology-f588d1b49e55>.

[10] GeeksforGeeks. (2020). Residual Networks (ResNet) - Deep Learning. [online] Available at: <https://www.geeksforgeeks.org/residual-networks-resnet-deep-learning/>.

[11] Deepchecks. (2023). What is DenseNet | Deepchecks. [online] Available at: https://www.deepchecks.com/glossary/densenet/.

[12] GeeksforGeeks. (2024). DenseNet Explained. [online] Available at: https://www.geeksforgeeks.org/densenet-explained/.

[13] Le, K. (2021). An overview of VGG16 and NiN models. [online] Medium. Available at: <https://lekhuyen.medium.com/an-overview-of-vgg16-and-nin-models-96e4bf398484>.

[14] GeeksforGeeks. (2024). DenseNet Explained. [online] Available at: <https://www.geeksforgeeks.org/densenet-explained/>.

[15] Indeed Editorial Team (2022) How to Measure Accuracy and Precision in 5 Steps, Indeed Career Guide. Available at: <https://www.indeed.com/career-advice/career-development/how-to-measure-accuracy>.

[16] Understanding Precision versus Recall: Strike the Right Balance for Effective Analysis (2024) Graphite Note. Available at: <https://graphite-note.com/precision-versus-recall-machine-learning/>.

[17] Kundu, R. (2022) F1 Score in Machine Learning: Intro & Calculation, www.v7labs.com. Available at: <https://www.v7labs.com/blog/f1-score-guide>.

‌

‌

‌

‌

‌

‌

‌

‌

‌

‌

‌

‌

‌