Godfrin Louis | Billy Romain
Bussiere Alexandre

AI

ST2AIM - Artificial Intelligence and ML for IT Engineer

Project

Professor : AIT EL MAHJOUB, Youssef

Table des matières

This document provides an explication of the AI project that we made, which is designed to implement and train various Convolutional Neural Networks on the CIFAR-10 dataset.

Overview

The code implements a deep learning framework using TensorFlow to build and train three different VGG models (VGG1, VGG2, VGG3) with different configurations and optimizers. The primary goal is to classify images from the CIFAR-10 dataset, which consists of 10 classes of objects, that are represented by airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck.

We also add functions to display training history, class distribution, and sample images.

Hyperparameters

max_epochs

max_epochs is set to 100. This value determines the maximum number of training epochs for the models, which is essential for controlling the training time and performance.

class_names

class_names is a list containing the names of the 10 classes in the CIFAR-10 dataset: ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']. This list is used for labeling the images and interpreting the model's predictions.

Learning_rate

The learning_rate is a parameter that determines the size of the step that the algorithm must make to adapt the model weights at each iteration during training.

Here the code for each optimizer:

```
"SGD": tf.keras.optimizers.SGD(learning_rate=0.05),

"Adam": tf.keras.optimizers.Adam(learning_rate=0.001),

"RMSprop": tf.keras.optimizers.RMSprop(learning_rate=0.0001)
```

Dropout

Dropout is a technique used to randomly and temporarily disable certain neurons in the network, as well as all its incoming and outgoing connections within a model layer during training. The goal of this operation is to prevent the model from overfitting (becoming too accustomed to the training set and thus failing to generalize well to new data).

We applied the following strategy, after each layer we used dropout progressively.

For example, in the VGG3 model, we drop 20% of the neurons in the first layer, 40% in the second, and 50% in the third:

```
def vgg3_with_regularization ():

vgg_block(2, 32),

tf.keras.layers.Dropout(0.2),  # Dropout layer

vgg_block(2, 64),

tf.keras.layers.Dropout(0.4),  # Dropout layer

vgg_block(2, 128),

tf.keras.layers.Dropout(0.5),  # Dropout layer
```

Functions

In this part we going to explain our code, with details on each function and the variables that we use inside.

If you need or want more information, go through the code on GitHub in main.py.

vgg_block(num_convs, num_filters)

This function creates a block of convolutional layers followed by a max pooling layer. It takes two parameters:

- **num_convs**: The number of convolutional layers in the block.
- **num_filters**: The number of filters to use in each convolutional layer.

The function utilizes ReLU activation, which handle non-linearity and batch normalization, is used for the training process and is activate through all the process until the output layer. After adding the convolutions, a max pooling layer is added to downsample the feature maps.

ReLU is applied after each convolutional layer, and batch normalization normalizes the output after each layer.

plot_sample_images(images, labels, classes)

This function visualizes a sample of images and their corresponding labels.

It takes three parameters:

- **images**: A batch of images to be displayed.
- **labels**: The labels corresponding to the images.
- **classes**: The list of class names for labeling the images.

It displays a 3x3 grid of images with their labels. This function helps verify if the model's predictions match the actual labels, allowing for a check of the classification accuracy.

plot_class_distribution(labels, class_names)

This function visualizes the distribution of classes in the dataset.

It takes two parameters:

- **labels**: The labels of the dataset.
- **class_names**: The list of names corresponding to the labels.

It generates a bar chart to display the frequency of each label based on the model's output.

plot_history(history, model_name, max_epochs)

This function plots the training history of the model.

It takes three parameters:

- **history**: The training history returned by the fit method.
- **model_name**: A string representing the name of the model.
- **max_epochs**: The maximum number of epochs used during training.

The function creates two subplots: one for accuracy and another for loss, created for a clear comparison between the training and real values, to ensure optimal adjustments to have the better performance over epochs.

vgg1_with_optimization()

This function builds the first VGG model with basic optimization, featuring:

- 2 convolutional layers with 32 filters, followed by a dropout layer to reduce overfitting.
- Flattening of the output and 2 dense layers with ReLU and softmax activations for classification.

It returns the constructed model.

vgg2_with_regularization()

This function builds the second VGG model, adding regularization techniques:

- 2 convolutional layers with 32 filters and another 2 with 64 filters, each followed by a dropout layer.
- As in VGG1, the output is flattened and passed through dense layers.

It returns the constructed model.

vgg3_with_regularization()

This function builds the third VGG model with more complexity:

- 2 convolutional layers with 32 filters, followed by 2 layers with 64 filters, and another 2 with 128 filters, each followed by dropout layers.
- Flattening and dense layers are included like the previous models.

It returns the constructed model.

compile_and_train(model, optimizer, x_train, y_train_one_hot, x_test, y_test_one_hot, max_epochs)

This function compiles and trains the model with the specified parameters:

- **model**: The model to be trained.
- **optimizer**: The optimizer to use during training.
- **x_train**: Training data.
- **y_train_one_hot**: One-hot encoded training labels.
- **x_test**: Testing data.
- **y_test_one_hot**: One-hot encoded testing labels.
- **max_epochs**: The maximum number of training epochs.

The function compiles the model using categorical crossentropy loss and accuracy metrics and then fits the model to the training data, returning the training history.

Loading and Preprocessing Data

The CIFAR-10 dataset is loaded and normalized to a range of [0, 1]. Labels are converted to one-hot encoding.

Training Models

Three models are instantiated and trained using different optimizers (SGD, Adam, RMSprop). The results are printed in the console for tracking progress.

Visualizations

After training, the model performance is visualized using the plot_history function, and the result of class distribution is plotted using plot_class_distribution. Sample images with their corresponding labels are displayed using plot_sample_images.

Interpretation

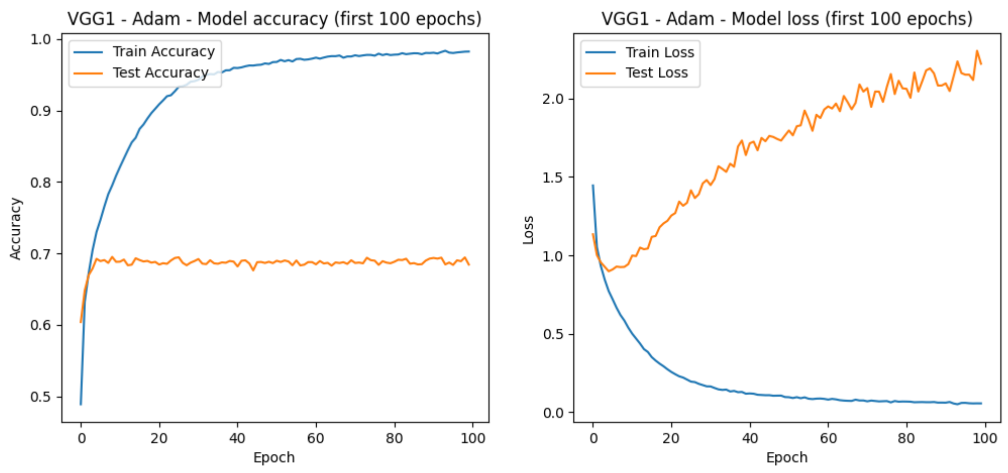
In this part we will see that more we add vgg_block, so layers, better are the result.

Adam

Adam (Adaptive Moment Estimation) is a widely used optimization algorithm in deep learning, it is effective for fast convergence and stable training.

VG1

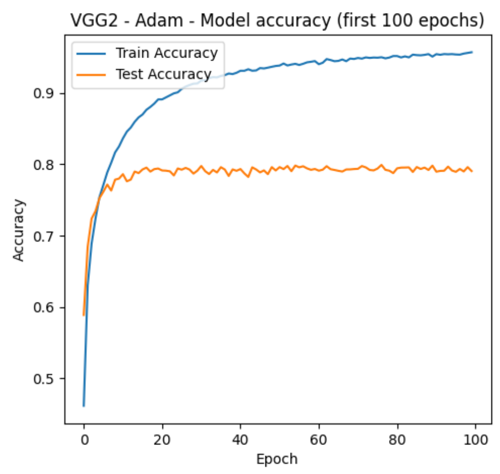
Max Training Accuracy	**Max True Accuracy**
98 %	**70 %**

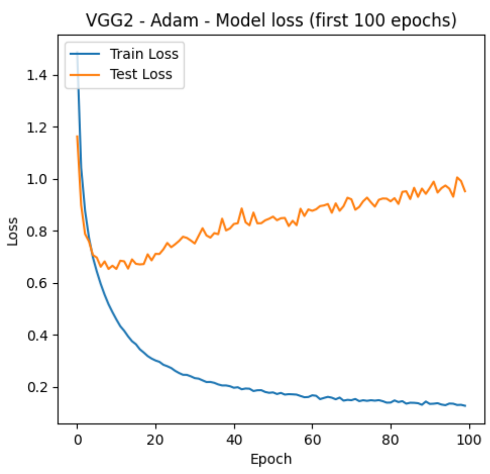


The training achieves high accuracy, but the test accuracy remains relatively low, indicating possible overfitting. This is corroborated by the loss graph, where the test loss increases, while the training loss decreases sharply.

VG2

Max Training Accuracy	Max True Accuracy
95 %	80 %

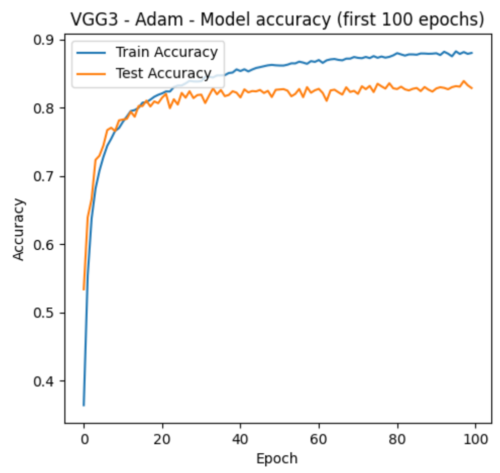


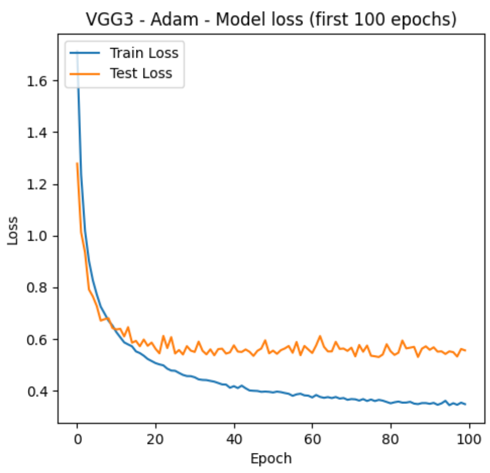


There has been an improvement in comparison to VGG1. The test accuracy is better, suggesting that the model generalises slightly better. The test loss is still higher than training loss but is more stable overall than in VGG1.

VG3

Max Training Accuracy	Max True Accuracy
91 %	84 %





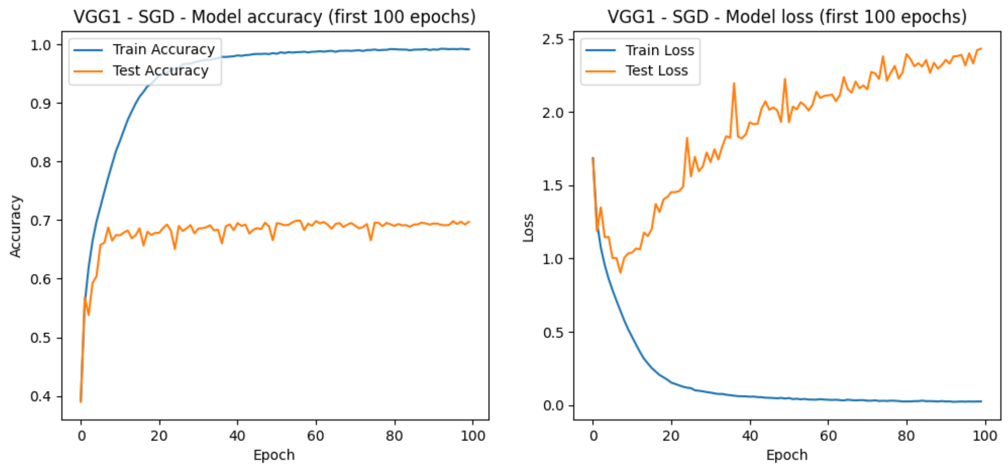
VGG3 shows a good balance between training accuracy and test accuracy, with values close to each other, indicating better generalisation of the model. The losses, training and test, also converge faster and stabilise at lower values, showing that the model is better optimised.

SGD

SGD (Stochastic Gradient Descent) is a basic but effective optimization algorithm in deep learning.
It updates model parameters using small, random batches of data, which helps reduce computation time and can escape local minima.

VG1

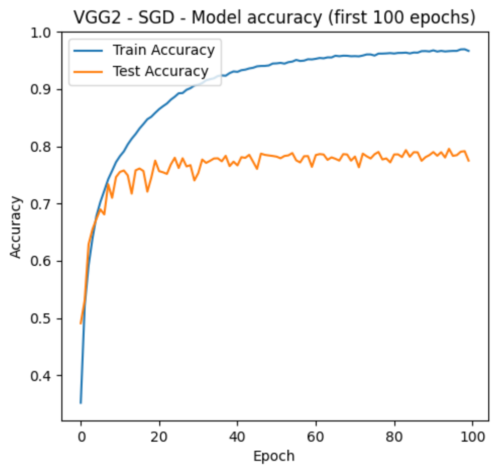
Max Training Accuracy	Max True Accuracy
99 %	69 %

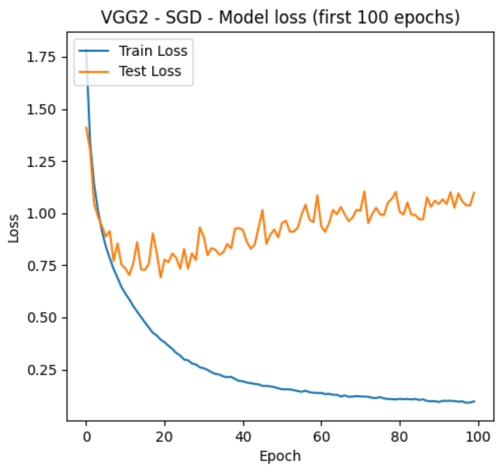


Although the training accuracy is high, the test accuracy remains low, indicating overfitting. The loss graph shows that the test loss increases significantly, confirming this problem.

VG2

Max Training Accuracy	Max True Accuracy
97 %	77 %

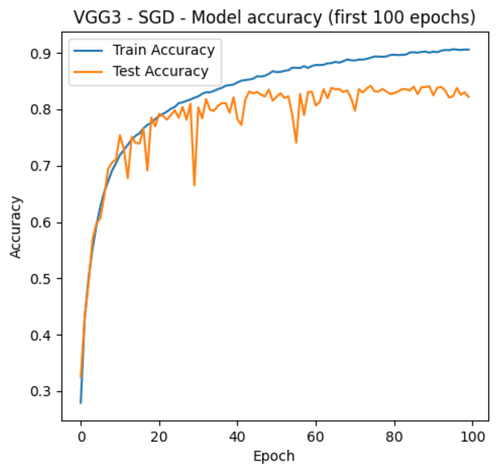


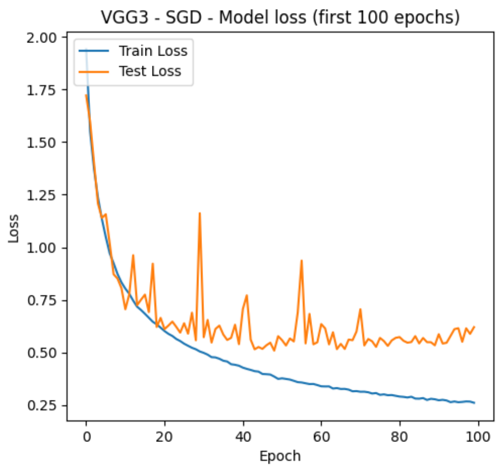


Compared with VGG1, VGG2 has better test accuracy, suggesting better generalisation. However, the test loss remains high and fluctuates, indicating that the model has to be adjusted or need to apply regularisation.

VG3

Max Training Accuracy	Max True Accuracy
90 %	82 %





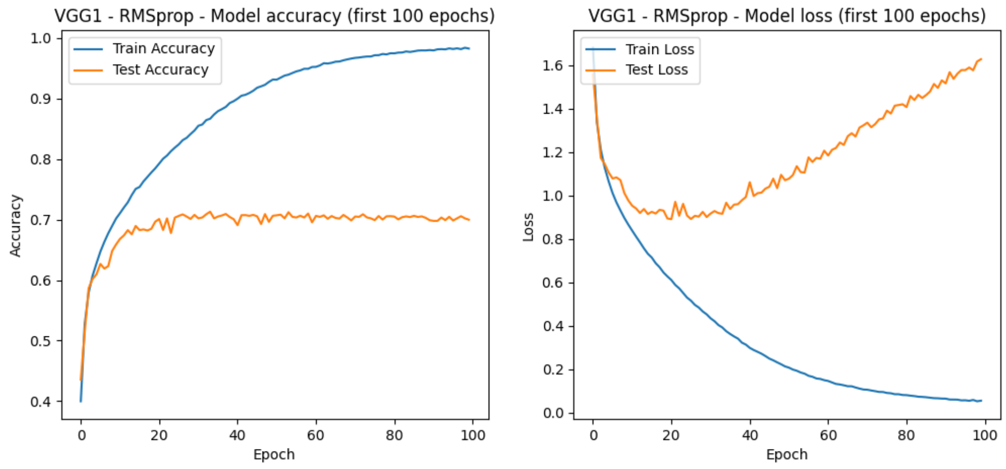
VGG3 shows a good compromise between train and test accuracy, with close values, indicating an improvement in generalisation. The losses, train and test, converge better compared with previous models, although the test loss shows some fluctuations.

RMSprop

RMSprop (Root Mean Square Propagation) is an adaptive learning rate optimization algorithm. It adjusts the learning rate for each parameter based on recent gradients.

VG1

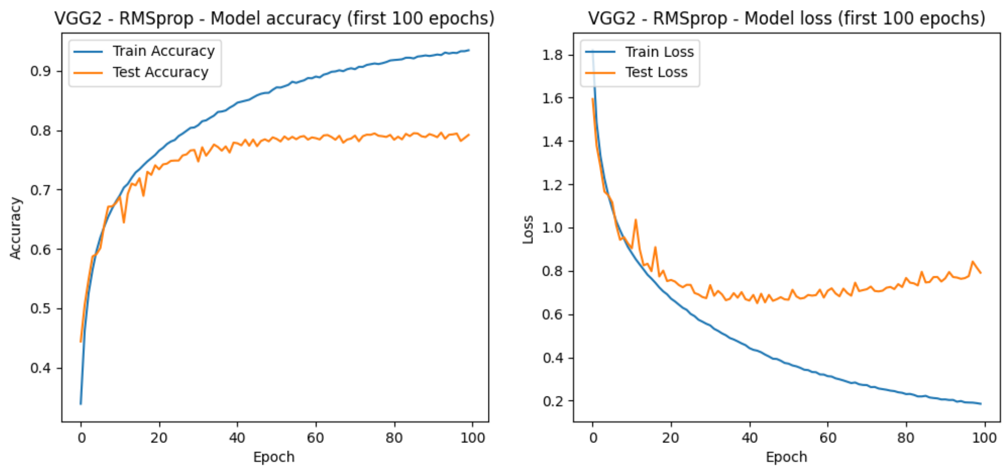
Max Training Accuracy	Max True Accuracy
97 %	**70 %**



The model shows high training accuracy, but test accuracy remains low, indicating overfitting. The loss graph shows that the test loss increases, confirming this phenomenon, while the training loss decreases significantly.

VG2

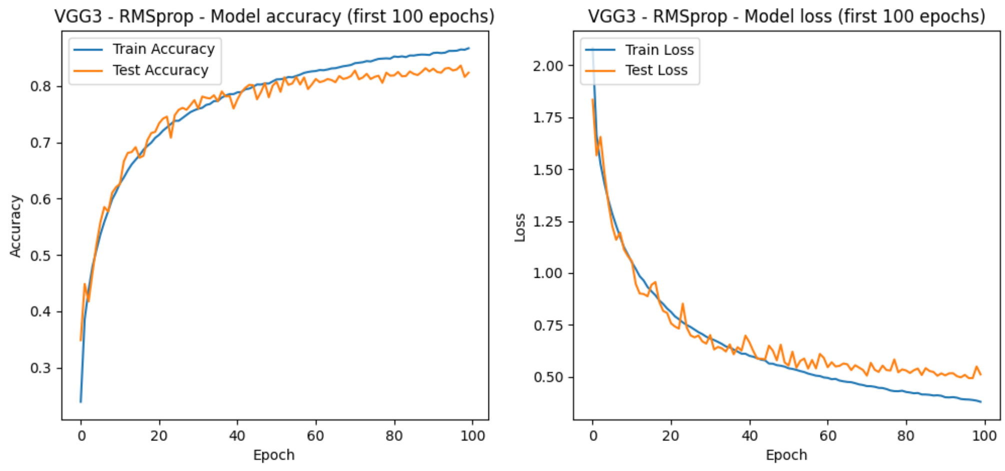
Training Accuracy	True Accuracy
93 %	77 %



Compared with VGG1, VGG2 showed an increase in test-phase accuracy, indicating an improvement in generalisation ability. However, the loss in the test phase remains higher than that in training.

VG3

Training Accuracy	True Accuracy
85 %	83 %



VGG3 shows good results between training and test accuracy, with values very close to each other, which mean a better generalisation of the model. The loss curves, training and test, converge more smoothly, demonstrating that the model is better optimised and regularised.

For all 3 we see that VG3G is obviously the best of the 3 in results with no consideration for training time.
Of the 3 different optimizer Adam outperforms 1 % the other, but as it is only a single epoch that managed to reach 84 %, and the other where also stuck at 83.9% for RMSprop and 83.1% for SGD. We can still presume that RMSprop and Adam are comparable, with a slight difference performance from Adam with is 84%.

For all the trained models, we observe a convergence of the loss value between the training loss and the test loss as the number of layers increases.
My hypothesis is that a model with fewer layers will be overfit much sooner than one with more layers, as it lacks the capacity to generalize beyond the training data. Because we see that for VG1, the deviation begins heavily around the 5th to 10th epoch; for VG2, it starts between the 10th and 15th epoch. VG3 remains relatively close, but we observe the beginning of a divergence between the 15th and 20th epoch, with a notable outlier: RMSprop, which starts to deviate around the 50th to 60th epoch.

Answers to the questions from the subject

Part I:

3. How many training and test samples are there in the CIFAR-10 dataset?

The CIFAR-10 contains 50,000 training images and 10,000 test images.

4. What is the shape of each image? How many color channels are there?

Each image is 32x32 pixels with 3 colour channels (RGB).

Part II:

2. Display the corresponding labels for the images. What type of data do the labels represent? Are they categorical or numerical ?

The labels are categorical, representing classes such as aeroplanes, automobiles and so on.

3. What do the images look like? Are they easy or difficult to classify based on human vision?

The images can be simple or difficult to classify, depending on the class and visual detail.

Part III:

2. Explain why normalization is important in Neural Networks models.

Normalisation enables neural networks to converge more quickly.

3. What are the benefits of normalizing the pixel values in an image dataset?

It improves accuracy and reduces errors.

4. How does normalization impact the performance of neural networks?

Effective normalisation stabilises and accelerates learning.

Part IV:

2. Why is it necessary to apply one-hot encoding to the class labels in a classification problem?

This allows labels to be treated as distinct categories in classification.

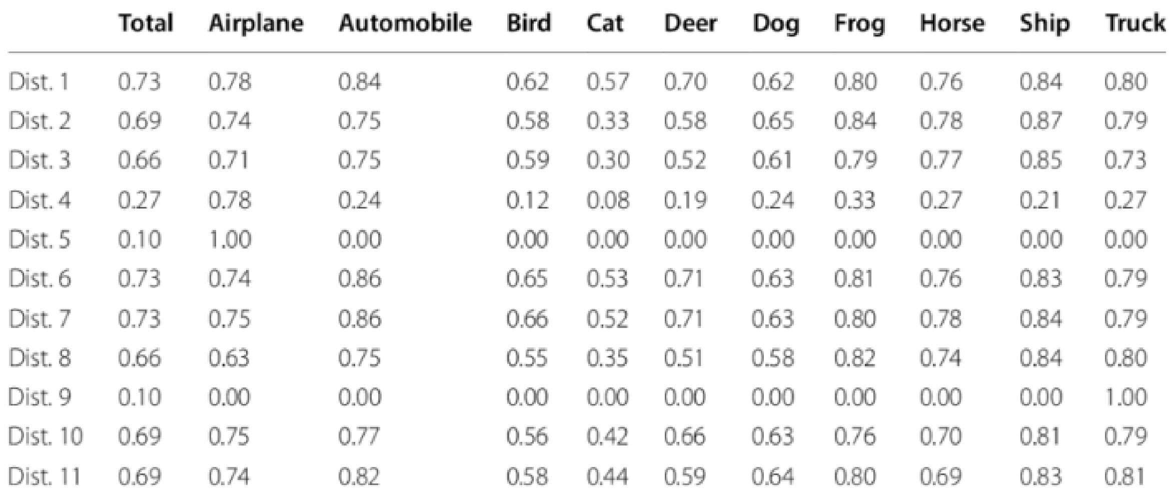
3. What does one-hot encoding represent, and how does it help in training a neural network?

Encoding helps optimise network learning.

Part V:

2. Are the classes balanced in the CIFAR-10 dataset, or are some classes over/underrepresented?

Yes, some classes are underrepresented, but it was complicated to tell how much it's unbalance by a graph, so we made a quick research and find this:



	Total	Airplane	Automobile	Bird	Cat	Deer	Dog	Frog	Horse	Ship	Truck
Dist. 1	0.73	0.78	0.84	0.62	0.57	0.70	0.62	0.80	0.76	0.84	0.80
Dist. 2	0.69	0.74	0.75	0.58	0.33	0.58	0.65	0.84	0.78	0.87	0.79
Dist. 3	0.66	0.71	0.75	0.59	0.30	0.52	0.61	0.79	0.77	0.85	0.73
Dist. 4	0.27	0.78	0.24	0.12	0.08	0.19	0.24	0.33	0.27	0.21	0.27
Dist. 5	0.10	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Dist. 6	0.73	0.74	0.86	0.65	0.53	0.71	0.63	0.81	0.76	0.83	0.79
Dist. 7	0.73	0.75	0.86	0.66	0.52	0.71	0.63	0.80	0.78	0.84	0.79
Dist. 8	0.66	0.63	0.75	0.55	0.35	0.51	0.58	0.82	0.74	0.84	0.80
Dist. 9	0.10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00
Dist. 10	0.69	0.75	0.77	0.56	0.42	0.66	0.63	0.76	0.70	0.81	0.79
Dist. 11	0.69	0.74	0.82	0.58	0.44	0.59	0.64	0.80	0.69	0.83	0.81

Imbalanced CIFAR-10 classification [79]

On this website: https://www.researchgate.net/figure/Imbalanced-CIFAR-10-classification-79_tbl1_332165523

So yes, it's an unbalance dataset.

3. Why is it important to analyze the class distribution before training a model?

Understanding the distribution helps to avoid bias in the model.

Optimizers

1. What is an optimizer in the context of deep learning?

An optimizer is an algorithm used to adjust the weights and biases of a neural network during training to minimize the loss function.

2. Explain the role of an optimizer during the training of a neural network. How does it affect the learning process?

The role of an optimizer during the training of a neural network is to adjust the model's parameters, weights and biases, to minimize the loss function. Which measures the difference between the model's predictions and the actual target values. Also, that the accuracy and performance of the model are the best it can be.

3. What are the most common types of optimizers?

We can say that the 3 used in this project are some of the most used but we alse have Adadelta / Adagrad or Adamax an upgrade for Adam that is used if the Weight gradient created is unusually big.

4. Compare how different optimizers (SGD, Adam, etc) update the model weights during training and how this affects learning efficiency and speed.

- **SGD** updates weights by computing the gradient of the loss function considering each parameter and moving in the opposite direction to minimize the loss. It uses a small random batch of data per update, which introduces noise and variability. It's unstable with random peaks and attain a plateau quite fast.

- **Adam** adapts the learning rate for each parameter based on the first and second moments of the gradients (the mean of the gradients and the uncentered variance of the gradients). This allows the optimizer to maintain a high learning rate while reducing it as the gradients become stable. It's stable and have a fast convergence, but plateau after a while.

- **RMSprop** scales the learning rate based on a moving average of recent gradients for each parameter. This helps manage the learning rate dynamically and avoids oscillations. Stable and consistently going down.

5. How do the different optimizers perform with the VGG architecture?

Optimizer	**Speed of Convergence**	**Stability**	**Generalization**	**Best VGG Architecture**
SGD	Slow	Prone to oscillations	Strong when tuned	VGG3 with **All**
Adam	Fast	Highly stable	May overfit if not regularized	VGG3 with **All** (best performance overall)
RMSprop	Moderate	Stable	Good, but can vary	VGG3 with **All**

All = Normalisation, dropout, better learning rate for the model, etc

6. Train your VGG model with different optimizers, analyze and compare their performance. Which optimizer performs best for the VGG architecture?

They are all quite close to each other, with an average accuracy of 83 to 84%. However, with a maximum of 100 epochs, Adam comes out as the winner, as it is the only one capable of reaching 84% accuracy, even though this happened only once. So, Adam performs better than the others by less than a single percent.

Bonus - Regulization: Dropout and Batch Normalization Layers

Dropout: Explain the concept of dropout in neural networks. Why is it used, and how does it help prevent overfitting? Describe how dropout is implemented during training and inference. What differences exist in the way dropout is applied in these two phases?

Concept of Dropout:

Dropout is a regularization technique used in neural networks to prevent overfitting. It involves randomly “dropping out” a little portion of neurons in a layer during each training iteration. This means that these neurons are temporarily ignored, along with their connections.

The idea is to prevent the network from becoming too dependent on specific neurons and their interactions.

Goal:
By randomly deactivating neurons during training, dropout prevents the network from becoming overly reliant on specific paths or features. This forces the network to learn redundant representations, improving its ability to generalize to unseen data.

It reduces inter-dependencies among neurons, promoting diversity in the learned features. As a result, the network becomes less sensitive to specific inputs, making it more robust.

Implementation During Inference:

During inference, dropout is not applied. Instead, all neurons remain active.

The network outputs are not scaled during inference. The scaling applied during training; this ensures that the network behaves consistently in both phases.

Differences Between Training and Inference:

Training: Neurons are randomly dropped based on a probability, and active neurons are scaled by 1/probability.

Inference: All neurons are active, and no scaling is applied. This ensures that the model's outputs during inference are as expected based on the training phase.

Batch Normalization: Define batch normalization. What are its main objectives, and how does it work? Discuss the benefits of batch normalization. How does it impact the training speed and stability of a neural network?

Definition of Batch Normalization:

Batch normalization is a technique used to normalize the activations of neurons within a mini-batch. It involves scaling, and shifting the inputs of each layer. So that they have a mean of zero and a variance of one. This is done during training.

2. Objectives of Batch Normalization:

By normalizing the inputs to each layer, BN reduces the internal covariate shift (i.e., changes in the distribution of layer inputs during training). This stabilization helps the network converge faster.

BN has a slight regularization effect, as each mini-batch is normalized based on its own statistics, introducing noise that makes the model more robust.

3. How Batch Normalization Works:

For each mini-batch during training, BN calculates the mean μ and variance $\sigma^2$of the inputs to a layer.

The inputs x are then normalized:

$$\hat{x} = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

BN introduces learnable parameters, γ\gammaγ (scale) and β\betaβ (shift), allowing the network to adjust the normalized values:

$$y = \gamma \cdot \hat{x} + \beta$$

This ensures that the network can learn optimal representations, even after normalization.

4. Benefits of Batch Normalization:

By stabilizing the input distributions, BN allows for higher learning rates and faster convergence. BN acts as a form of regularization, reducing the risk of overfitting, especially when used with other techniques like dropout.

The normalization process makes the network less dependent on specific weight initializations, improving the overall robustness of the training process.

5. Impact on Training Speed and Stability:

BN significantly accelerates training by reducing the internal covariate shift, which allows for higher learning rates and faster convergence.

BN reduces the likelihood of gradients exploding or vanishing, making the network more stable during training. This stability helps deeper networks converge more effectively and allows them to generalize better to new data.