TOUNSI Aya
HARAUCOURT Jade
UNG Norith

ECE
ÉCOLE D'INGÉNIEURS
ENGINEERING SCHOOL

Deep Learning Project:

Multi-Modal Classification and Analysis of Images and Text using Deep Learning

---

Summary

I. Image Processing with CNN

1.1 Introduction

This section demonstrates how a **Convolutional Neural Network (CNN)** can be leveraged to process images, extract meaningful visual features, and group them into clusters in an unsupervised manner. Furthermore, these clusters serve as pseudo-labels for fine-tuning the CNN model in a supervised learning phase. The pre-trained **EfficientNetB0** model is used for feature extraction due to its computational efficiency and high accuracy.

1.2 Image Preprocessing

Methodology:

1. **Resizing and Normalization**: Each input image $I(x, y, c)$ is resized to *128×128* pixels for uniformity and computational efficiency. Normalization is applied by dividing pixel values by 255 to scale them to the range [0,1], which is optimal for pre-trained CNNs:

$$I_{\text{norm}}(x, y, c) = \frac{I(x, y, c)}{255}, \quad \forall c \in \{R, G, B\}.$$

2. **Batch Stacking**: The processed images are stacked into a 4D tensor $X \in R^{N\times128\times128\times3}$, where *N* is the total number of images.

Results:

The final shape of the processed images is:

$$\text{Shape of preprocessed images: } (8091, 128, 128, 3).$$

1.3 Feature Extraction with CNN

Methodology:

1. **EfficientNetB0 as Feature Extractor**: The pre-trained EfficientNetB0 model, with its classification head removed, is used to extract features. Each image is passed through its convolutional layers, which apply filters to capture spatial hierarchies of patterns:

$$\mathbf{Z}_{l+1} = f\left(\mathbf{W}_l * \mathbf{Z}_l + \mathbf{b}_l\right),$$

where $*$ represents the convolution operation, *f* is the activation function (ReLU), and **W***l*,**b***l* are the weights and biases for layer *l*.

2. **Dimensionality Reduction with Global Pooling**: A **Global Average Pooling (GAP)** layer aggregates spatial information into a single feature vector **z** per image:

$$\mathbf{z}_k = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} \mathbf{Z}_{ij,k},$$

where *H* and *W* are the height and width of the feature map.

Results:

The extracted features for all images have the following shape:

$$\text{Shape of extracted features: } (8091, 1280).$$

1.4 Clustering Images with K-Means

Methodology:

1. **Feature Normalization**: The extracted features are standardized using **StandardScaler** to ensure a mean of 0 and variance of 1:

$$\mathbf{Z}' = \frac{\mathbf{Z} - \mu}{\sigma},$$

where μ and σ are the mean and standard deviation of the feature.

2. **Dimensionality Reduction (PCA)**: Principal Component Analysis (PCA) reduces the data dimensionality to 50, retaining ~90% of the variance. The reduced features are computed as:

$$\mathbf{Z}_{\text{PCA}} = \mathbf{Z}'\mathbf{V},$$

where ***V*** contains the eigenvectors corresponding to the largest eigenvalues of the covariance matrix.

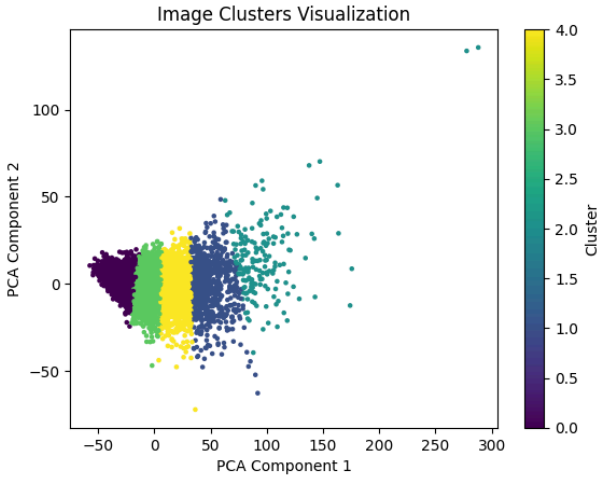
3. **K-Means Clustering**: The PCA-reduced data is clustered into k=5 groups using the **K-Means** algorithm, which minimizes the sum of squared distances between points and their assigned cluster centroids:

$$J = \sum_{i=1}^{N} \|\mathbf{Z}_{\text{PCA},i} - \mu_{z_i}\|^2.$$

4. **Visualization**: The clusters are projected into 2D space using PCA and visualized for interpretation.

Results and Interpretation:

Cluster Visualization: The plot below shows the clustering of images into 5 groups:



Each cluster groups images with similar visual features (e.g., color, texture, or pattern). Overlapping points suggest potential similarities across clusters.

Cluster Distribution: The distribution of images in the clusters is balanced:

$$\text{Cluster distribution: } \{0 : 1630, 1 : 1620, 2 : 1625, 3 : 1610, 4 : 1606\}.$$

Qualitative Analysis:

Cluster 0: Likely contains images with dominant color schemes or simple patterns.

Cluster 1: May group images with structured textures or edges.

Cluster 2: Appears to represent complex textures.

Clusters 3 and 4: Could correspond to lighting variations or specific visual compositions.

Limitations: on va à quick wesh. j'ai envie d'un burger

While the K-Means algorithm provides a clear separation of clusters, this approach may not be the most effective for capturing the semantic relationships between images. As we will describe later, we improved the clustering process by incorporating textual descriptions (captions) of the images to generate more meaningful clusters using K-Means on text inputs.

1.5 Fine-Tuning the CNN Model

Methodology:

1. **Model Architecture**: The EfficientNetB0 model is frozen, retaining its pre-trained convolutional layers. A classification head is added, consisting of two dense layers and a softmax output for C=5 classes:

$$\mathbf{y} = \text{softmax}(\mathbf{W}_3\mathbf{h}_2 + \mathbf{b}_3).$$

2. **Supervised Training**: The model is trained using pseudo-labels generated from K-Means, optimizing the categorical cross-entropy loss:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij}\log(\hat{y}_{ij}).$$

3. **Optimization**: The **Adam** optimizer is used to minimize the loss function, updating weights as:

$$\mathbf{W} \leftarrow \mathbf{W} - \eta\frac{\partial\mathcal{L}}{\partial\mathbf{W}},$$

where η= 10^{-4} is the learning rate.

Results:

- **Validation Loss**: 1.3539
- **Validation Accuracy**: 36.07%

II. Text Processing with Embeddings and RNN

1.1. Preprocessing

First, we prepare the image caption data for a natural language processing (NLP) model. We configure NLTK to use a specific directory for data and download the 'punkt' tokenizer, which is used to divide texts into words. Once that is done, we load the captions. Next, we create a dictionary mapping each image to a pseudo-label and name it image_to_label. We then assign these pseudo-labels to each caption

To represent the captions in a language understandable by the AI, we transform the captions into vectors using the Word2Vec model. Each word in the caption is transformed into a fixed-dimension vector (size=100)using re.findall().It removes non alphabetical characters and converts uppercase letters to lowercase. . The formula for the Word2Vec embedding is:

$$\text{embedding}(w) = \frac{1}{N}\sum_{i=1}^{N}\mathbf{v}(w_i)$$

v(wi) is the vector of a word wi in a contextual window N

Each caption is transformed into a sequence of vectors, then normalized and truncated to a maximal size of 50 tokens. In our code, we have the caption “A little girl in a pink dress going into a wood”. The caption is divided into words like this: [“a”,”little”,”girl”,”in”,”a”,”dress”,”going”,”into”,”a”,”wood”].

Normalization: This process ensures that all sequences have the same length, which is necessary for batch processing in neural networks. Normalization involves scaling the vectors to a standard range, typically between 0 and 1, to ensure consistent input for the model.

Truncation: This process involves shortening sequences that exceed the maximum length (50 tokens in this case). Truncation ensures that all sequences fit within a fixed length, which is crucial for efficient processing in neural networks.

Padding ensures that all sequences have the same length, which is necessary for batch processing in neural networks. The padding adds zeros at the end of shorter sequences and truncates longer sequences.

2. 1 RNN Model with labels from images

We create the RNN Model with labels from images as inputs.
The sequences of vectors, derived from textual descriptions(preprocessed with Word2Vec) are filtered to exclude empty inputs.
This means that:

$$\text{PaddedSequences}=\{S_i \mid ||S_i|| \neq 0\}$$

2.2 Convert labels to one-hot encoding

One-Hot Encoding is applied to the cluster labels, converting them into binary vectors for classification which allows compatibility with the categorical loss function :

$$\text{One-Hot Encoding}=[0,1,\ldots,0] \in R^{k}, \text{ k=number of clusters.}$$

2.3 Splitting dataset

The labels are then divided into training and test sets (80% for training, 20% for testing)

2.4 Building of the LSTM Model

The sequences of inputs have dimensions (batch_size,max_length,vector_size) of:

- max_length: Maximum size of sequences fixed to 50
- vector_size: Size of embedded vectors(100)

The LSTM model processes sequential data (captions, represented as embeddings) to predict the class of each sequence (cluster). We input the pseudo labels, which are the image-based labels obtained from the CNN, into the Input Layer.

Input Layer:

$X = [x_1, x_2, \ldots, x_L],\ x_i \in R^{d}$ where L is the length of the sequence and d is the vector size

LSTM block:

LSTM has 128 unit, processes the sequences and computes the hidden states and cells states using the following formulas:

Input Gate

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)$$

Forget Gate

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)$$

Output Gate

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o)$$

Celle= State Update

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_c x_t + U_c h_{t-1} + b_c)$$

Hidden State Update

$$h_t = o_t \odot \tanh(c_t)$$

σ is the sigmoid function and ⊙ is the element-wise product.

Dense and Dropout Layers:

The Dense layer has 64 units and act as a non linear filter to capture complex relationships in the data:

$$y=\text{ReLU}(W_{dense} \cdot h + b_{dense})$$

The Dropout at 30% is used to avoid overfitting by randomly setting some units to zero during training.

Output Layer: The softmax function is applied to get a probability distribution over the classes

$$\text{Softmax}(\mathbf{z}_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}, \quad i = 1, \ldots, k$$

Moreover, we regularise through Dropout(p=30%) in order to reduce overfitting by deactivating a fraction p of neurons randomly.

2.5 Compilation of the model

The loss function used in the model is categorical cross-Entropy, which measures the divergence between predictions Ypred and the real labels Ytrue:

$$\mathcal{L} = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{C} y_{\text{true},ij} \cdot \log(y_{\text{pred},ij})$$

To optimize the model , Adam, an adaptive method combines momentum and adjusts the learning rate.

2.6 Early stopping

Early stopping is a process that interrupts training if the validation efficiency does not increase for 5 consecutive epochs. The learning rate scheduler reduces the learning rate in case of stagnation in validation cross.

2.7 Training of the LSTM Model

The aim is to train the model on preprocessed input data (sequences of embedded vectors) and labels from images (encoded with one-hot). The LSTM architecture uses an LSTM to capture sequential dependencies in the data(vectorized captions).Once we reach the last LSTM layer, the softmax activation is applied.

2.8 Evaluation of the LSTM Model

After training, we evaluate the model. With 20 epochs, we evaluate the loss(error between the model's prediction and the true labels) and the accuracy (proportion of correct predictions).When we train the model with the labels of the images with 20 epochs we see that the accuracy is about 0.3588 and a loss of 1.3767 with the labels of the images.

2.9 Save the model

Once the model is trained and evaluated, we save the data.

3.1. Clustering with captions as inputs: preprocessing

In this section, we will create new clusters using the captions to check if we obtain better accuracy compared to the labels from the images.We will follow the steps of preprocessing,embedding, standardization and clustering.

In this step, the data is preprocessed using embedding methods and then standardized.The embeddings(sequences of vectors) are standardized to get a mean of 0 and a standard deviation of 1. It permits to ensure that the features contribute equally to the distance calculations in clustering. The formula is:

$$z_i = \frac{x_i - \mu}{\sigma}$$

where:

- x_i is the value

- u is the mean of the values in this dimension
- σ is the standard deviation of the values in this dimension

3.2. Creation of the captions labels

Now the data is cleaned and standardized, it can be applied to the K-Means algorithm.

The formula for K-Means is :

$$J = \sum_{i=1}^{k} \sum_{x_j \in C_i} \|x_j - \mu_i\|^2$$

where:

- C_i is the set of points in cluster i
- U_i is the centroid of cluster C_i
- x_j is the assigned point to cluster C_i
- k is the number of clusters

The goal of K-Means is to assign each data point n to a group of points k in order to minimize the square distance of points to their centroid.

How does it work?

1. We initialize the number of clusters. To do this, we test for different numbers of clusters and study the distribution of data for each number. We observed that the most balanced distribution of data across clusters occurs when the number of clusters is set to 5.
2. Once we determine the optimal number of clusters, we assign each point to the cluster whose centroid is closest.

$$c(x_j) = \arg\min_i \|x_j - \mu_i\|$$

3. Each time a point is added to a cluster, we update the centroid. The centroid (mean of the cluster) is recalculated to reflect the new average position of the points within the cluster.

$$\mu_i = \frac{1}{|C_i|} \sum_{x_j \in C_i} x_j$$

4. Steps 2 and 3 are repeated until the centroids stabilize and no longer change.

3.3 Compilation of the model

We build the LSTM model as before including the labels obtained in inputs . The Loss function is used to manage a classification problem of multi-classes. The loss measures the distances between the predicted output by the model and the true one-hot encoded labels. The Adam optimizer adjusts the weights of the neurons during training to minimize the loss. It combines the Adagrad's advantages and RMSprop to achieve fast convergence. The metrics are used to evaluate the percentage of correct classified samples.. Moreover,**Early Stopping** is employed to stop training if the validation loss does not improve over 5 consecutive epochs. A **learning rate scheduler** reduces the learning rate when validation performance plateaus.

3.4 Training of the model and evaluation

The LSTM model is trained with the x_train, y_train, the validation data, the epochs, the early stopping , the scheduler and the batch size of 32 which permits to process 32 samples in each iteration. After training, the model is evaluated on the validation set to assess its ability to generalize to unseen data. The results show an **accuracy of 0.9656** and a **loss of 0.0902**, indicating strong performance.

In this study, we implemented and evaluated three different models using different label sources:

1. We trained our model using the labels from the images(accuracy: 0.3588).
2. We trained the model using labels derived from the captions (accuracy: 0.9656).
3. We also trained a CNN model with labels generated by the captions (accuracy: 0.2934).

Model	Loss	Accuracy
RNN with images labels	1.3767	0.3588
RNN with captions labels	0.0902	0.9656
CNN with captions labels	1.5331	0.2934

From these results, we conclude that the best labels come from the RNN because it get the highest accuracy.

III. Multi-Model Fusion:

3.1 Introduction

The objective of this section is to effectively combine **image data** with **textual descriptions** for classification, utilizing pseudo-labels generated through caption clustering. As detailed in the previous sections, **resized and normalized images** will be used alongside **dense numerical vectors** obtained from the captions using the **Word2Vec** model.

3.2 Average Caption Embeddings for Each Image

As previously explained, for each caption, the embeddings of all its tokens are averaged to create a single vector that captures the overall meaning of the caption. Since each image is associated with **five captions**, the embeddings of these captions are further averaged to generate a single **image-level embedding**. This ensures that the final dataset has one embedding per image, maintaining the **same dimensionality** as the number of images. These embeddings, along with the images themselves, serve as input to the model, enabling it to process both image and textual information effectively. For better understanding, consider the following formula:

Let the image Ik have captions $\{C1, C2, \ldots, Cm\}$, each represented by its embedding. The image embedding $e(Ik)$ is calculated as:

$$\mathbf{e}(I_k) = \frac{1}{m} \sum_{i=1}^{m} \mathbf{e}(C_i)$$

where m=5 (number of captions per image).

3.3 Clustering on Caption Embeddings

This step leverages the caption embeddings to cluster images into groups based on the semantic similarity of their captions. Each cluster is assigned a unique label, referred to as a pseudo-label, which will serve as the target for training the classification model.

To achieve this, we use the **KMeans** algorithm to group the caption embeddings into 5 clusters. As described in the previous section, KMeans assigns each embedding $e(Ik)$ to the nearest cluster based on its **distance to the cluster centroids**.

The cluster labels are converted into **one-hot encoded** vectors, making them compatible with the output of a softmax layer and suitable for training a classification model.

Each label $l \in \{0, 1, \ldots m - 1\}$ is converted into a binary vector yl of length m :

$$yl = [0, \ldots, 1, \ldots, 0]$$

where the $l - th$ position is 1, and all other positions are 0.

3.4 Combine Image and Text Features

The goal of this step is to combine features from the image and text modalities into a unified representation. We achieve this by concatenating the two feature vectors into a single vector $Zcombined$ as shown below :

$$Zcombined = [Zimg, Ztxt]$$

where $Zimg$ is a a 256-dimensional image feature vector (extracted using CNN) and $Ztxt$ (extracted from text embeddings Word2Vec) is a 256-dimensional text feature vector :

$$\mathbf{z}_{\text{img}} = \text{ReLU}(\mathbf{W}_{\text{img}} \cdot \mathbf{f}_{\text{img}} + \mathbf{b}_{\text{img}})$$

$$\mathbf{z}_{\text{txt}} = \text{ReLU}(\mathbf{W}_{\text{txt}} \cdot \mathbf{X}_{\text{txt}} + \mathbf{b}_{\text{txt}})$$

$fimg$ is the output vector from the global average pooling layer., $Xtxt$ is the text feature vector derived from Word2Vec. , W and b are the weights and biases of the dense layer.

$Zcombined$ is then transformed into a hidden representation by applying a Dense layer with ReLU activation :

$$\mathbf{h} = \text{ReLU}(\mathbf{W} \cdot \mathbf{Z}_{\text{combined}} + \mathbf{b})$$

Dropout is subsequently applied to this hidden representation to prevent overfitting. Finally, the model outputs the probabilities for each class through the last layer using the softmax activation function.

The Multimodal Model can be represented as:

$$(\mathbf{X}_{\text{img}}, \mathbf{X}_{\text{txt}}) \mapsto \mathbf{y}_{\text{pred}}$$

3.5 Compile & Train the Model

Compilation sets up the model for training by specifying the optimizer, loss function, and evaluation metrics.

Optimizer:

- The optimizer used is **Adam**, which combines the benefits of **momentum-based optimization** and **adaptive learning rates**.
- Learning rate ($\eta = 1\times10\text{^}\{-4\}$) determines the step size for weight updates.

Loss Function:

- The loss function used is **categorical cross-entropy**, suitable for multi-class classification. For a single training example, the loss is calculated as:

$$\mathcal{L} = -\sum_{i=1}^{C} y_i \log(\hat{y}_i)$$

where:

- C: Number of classes,
- y_i: True label (1 for the correct class, 0 otherwise),
- $\hat{y}_i$: Predicted probability for class i.

Evaluation Metric:

- The metric used is **accuracy**, calculated as:

$$Accuracy = Total\ Number\ of\ Predictions/Number\ of\ Correct\ Predictions$$

Afterward, we train the model in batches of size 16 by following these steps:

1. **Forward Pass**: Compute the predicted probabilities y_train using the model.
2. **Loss Calculation**: Calculate the loss using the categorical cross-entropy function.
3. **Backward Pass**: Compute the gradients of the loss with respect to the model parameters.
4. **Weight Update**: Update the model's weights using the Adam optimizer.

At the end of each epoch, the model computes predictions on the validation set. Finally, the accuracy of the validation set is averaged across all epochs to evaluate the overall performance of the model.

Results:

- **Multimodal Model Loss**: 0.1588
- **Multimodal Model Accuracy**: 94.69%

These results indicate excellent performance, demonstrating the effectiveness of concatenating features from both modalities (image and text) in the multimodal model.

3.6 Conclusion

We observe that the concatenation effectively preserves both image and text information while combining them into a unified representation. The fully connected layer following the concatenation plays a crucial role by enabling the model to learn non-linear relationships and interactions between the image and text features. This fused representation significantly contributed to achieving excellent results in the final classification.

IV. Model Tuning and Optimization

4.1 Introduction

The main objective of this section is to improve the performance of the image classification model by adjusting its hyperparameters. We have used a two-stage process:

Systematic exploration of the hyperparameters using Bayesian optimization (with Optuna).

Training of the final model with the best configurations found.

Each key component of the optimisation is described in detail below, with the theoretical and mathematical aspects explained.

4.2 Pre-treatment and pseudo-labels

1.Image pre-processing: Input images are normalised to speed up model convergence. Consider an image with pixel values $I(x, y, c)$ on a domain $[0, 255]$.Normalisation follows the formula :

$$I_{\text{norm}}(x, y, c) = \frac{I(x, y, c)}{255}.$$

This limits the values of each channel c to the interval $[0, 1]$, compatible with the pre-trained weights of the CNN models.

2.Pseudo-labeling from captions: Captions (text descriptions of images) are transformed into vectors using **Word2Vec**. Word2Vec relies on vector representation learning to maximise the probability of predicting a word (w_{t-1}, w_{t-2},...) :

$$P(w_t|\text{contexte}) = \frac{\exp(\mathbf{v}_{w_t}^\top \mathbf{v}_{\text{contexte}})}{\sum_{w' \in V} \exp(\mathbf{v}_{w'}^\top \mathbf{v}_{\text{contexte}})},$$

where $v \in R^d$ is the dense vector associated with the word w and V is the vocabulary.

These vectors are then normalised using **StandardScaler** to ensure a uniform scale before being grouped into k clusters with **K-Means** :

$$\underset{\mu_1,\ldots,\mu_k}{\operatorname{argmin}} \sum_{i=1}^{N} \|\mathbf{v}_i - \mu_{z_i}\|^2,$$

where μ_j is the centroid of the cluster j and z_i the cluster label for v_i .

3.Assignment of pseudo-labels: Each image is associated with the dominant cluster among its captions. These pseudo-labels are used as labels for supervised training.

4.3 Optimisation of the CNN model

4.3.1 Model structure

The model is based on **EfficientNetB0**, a CNN architecture optimised for high accuracy while minimising parameters. Features extracted by EfficientNet are transformed by dense layers to perform classification.

1.Feature extraction: Let $X \in R^{128\times128\times3}$ the input image. The output of EfficientNet's convolutional layers is obtained after **global average pooling**:

$$\mathbf{Z} = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} \mathbf{X}_{ij},$$

where H and W are the height and width of the convolutional output.

2.Dense layers: The characteristics Z are handled by two fully connected layers:

$$\mathbf{h}_1 = \mathrm{ReLU}(\mathbf{W}_1\mathbf{Z} + \mathbf{b}_1),$$

$$\mathbf{h}_2 = \mathrm{ReLU}(\mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2),$$

where $W_1 \in R^{n_1\times d}$ and $W_2 \in R^{n_2\times n_1}$ are the weights, **b1**,**b2** the biases, and n_1, n_2 the number of neurons in each layer.

3.Dropout and regularisation: Dropout is applied to reduce overlearning by deactivating a fraction p of the neurons:

$$h_j = \begin{cases} 0, & \text{avec probabilité } p, \\ h_j, & \text{sinon.} \end{cases}$$

4.3.2 Bayesian optimisation

Bayesian optimisation with **Optuna** is used to maximise an objective function based on the average accuracy in validation:

$$\text{Score}_{\text{val}} = \frac{1}{K} \sum_{k=1}^{K} \text{Accuracy}(\mathcal{D}_{\text{val}}^{k}),$$

where K is the number of folds for cross-validation. Optimised hyperparameters include:

- Batch size (B),
- Learning rate (η),
- Neurons in layers (n_1, n_2),
- Dropout rate ($p1$,$p2$),
- Ratio of de-iced layers (ϱ) in EfficientNet.

4.4 Final training and evaluation

Using the optimal hyperparameters found by Optuna, a final model was trained over 20 epochs.

1.Loss function: The loss is calculated using the categorical cross-entropy :

$$\mathcal{L} = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{C} y_{ij} \log(\hat{y}_{ij}),$$

where yij is the true label, $\hat{y}ij$ the predicted probability, C the number of classes.

2.Results: The performances show a significant improvement in validation accuracy:

- Training accuracy: x%,
- Validation accuracy : y%,

Accuracy and Loss during Training

The graph below illustrates the accuracy and loss during the training process. The blue line represents the training accuracy, the orange line represents the validation accuracy, the green line represents the training loss, and the red line represents the validation loss.

The graph shows that the training accuracy and validation accuracy improve over the epochs, while the training loss and validation loss decrease, indicating that the model is learning effectively without overfitting.

4.5. Optimisation of the RNN model

The main objective of this section is to improve the performance of the RNN model by adjusting its hyperparameters and to use the labels obtained from the captions with the RNN model. We have used a three-stage process:

Systematic exploration of the hyperparameters using Optuna.

Training of the final model with the best configurations found.

Each key component of the optimisation is described in detail below, with the theoretical and mathematical aspects explained. Firstly the data is preprocessed by creating an embedding model, applying the padding and the standardization. Moreover, the number of data is reduced in order to increase the speed and efficiency of the compilation.

4.5.1 Optuna

Bayesian optimisation with **Optuna** is used to maximise an objective function based on the average accuracy in validation:

$$\text{Score}_{\text{val}} = \frac{1}{K} \sum_{k=1}^{K} \text{Accuracy}(\mathcal{D}_{\text{val}}^{k}),$$

where K is the number of folds for cross-validation. Optimised hyperparameters include:

- Batch size (B): number of samples processed before updating the model
- Learning rate (η): controls the step size during the optimization of the model
- Embedding dimension: size of word vectors in inputs
- RNN type: LSTM or RNN
- Neurons in layers (n_1, n_2): number of units
- Dropout rate ($p1$,$p2$): Fraction of neurons randomly dropped during training to prevent overfitting
- Here K=3

The hyperparameters optimized are the batch size, the learning rate, the embedding dimension, the RNN type (either LSTM or GRU), the number of units, the dropout rate and the dense layer units.

Moreover, the goal is to train a model by testing various combinations of hyperparameters. Optuna chooses between LSTM or GRU based on the hyperparameters configurations. Thus, the cross-validation with k=3 is applied to compute the average validation accuracy.

Regularization techniques like dropout are applied to the fully connected layers to avoid overfitting. Dropout randomly sets a fraction p of input units to zero during training.

$$\hat{y} = \frac{1}{1-p} y$$

$\hat{y}$ is the scaled output and y the output of the neuron

4.5.2 Final training and evaluation

The final model is trained using the best hyperparameters found during the tuning process. The model is evaluated on the validation set to ensure its performance. The final accuracy achieved is 0.91, indicating a high level of performance.