ECE 657 Assignment 3 Report
Problem 1

Using your preferred machine learning library, train a small convolutional network (CNN) to classify images from the CIFAR10 dataset. Note that most libraries have utility functions to download and load this dataset (TensorFlow, PyTorch, keras)

1. **Any preprocessing steps you made.**

The CIFAR-10 dataset has been preprocessed for training a machine learning model. This involves normalizing the pixel values of the images to fall within the range of [0, 1], a critical step to facilitate smoother convergence during training. The class labels have been one-hot encoded, enabling the model to handle multi-class classification tasks effectively. Furthermore, the dataset has been divided into separate training and validation sets to evaluate the model's performance and mitigate the risk of overfitting.

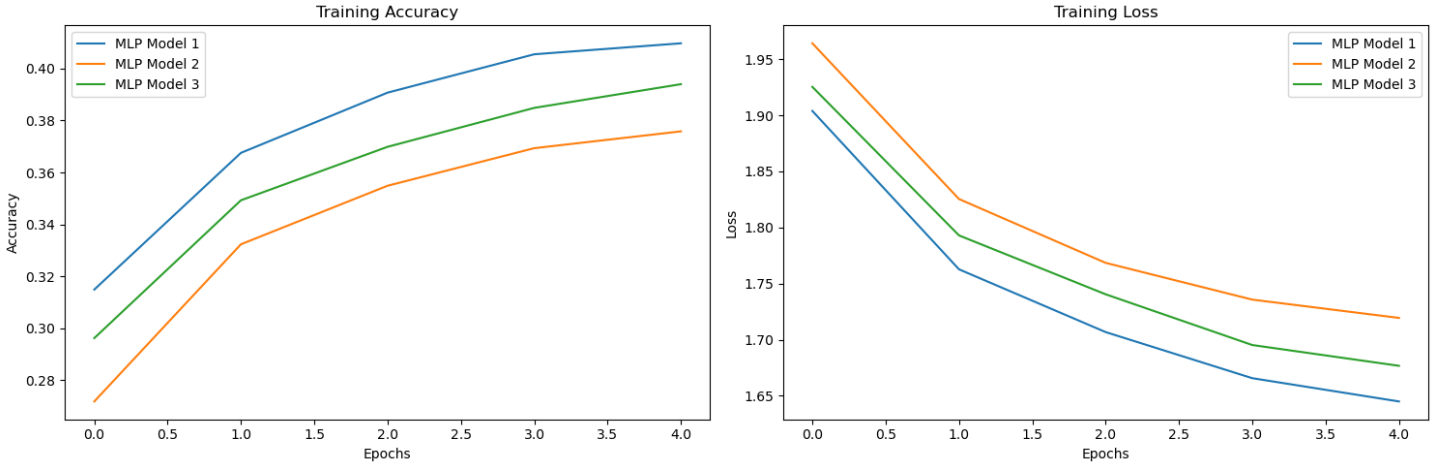
2. **Description of the output layer used and the loss function (use 1 setting for all 3 networks) and why you made these choices.**

In all three networks, the output layer utilizes the softmax activation function for multi-class classification. The loss function employed is categorical cross-entropy, chosen to optimize the model's ability to predict the correct class probabilities for the CIFAR-10 dataset, where one-hot encoded labels are used. These choices ensure the network produces valid probabilities and learns effectively to classify images into the appropriate classes.

3. **Change the number of layers and the number of neurons per layer in the MLP, plot/tabulate the training and validation accuracies and comment on the results.**

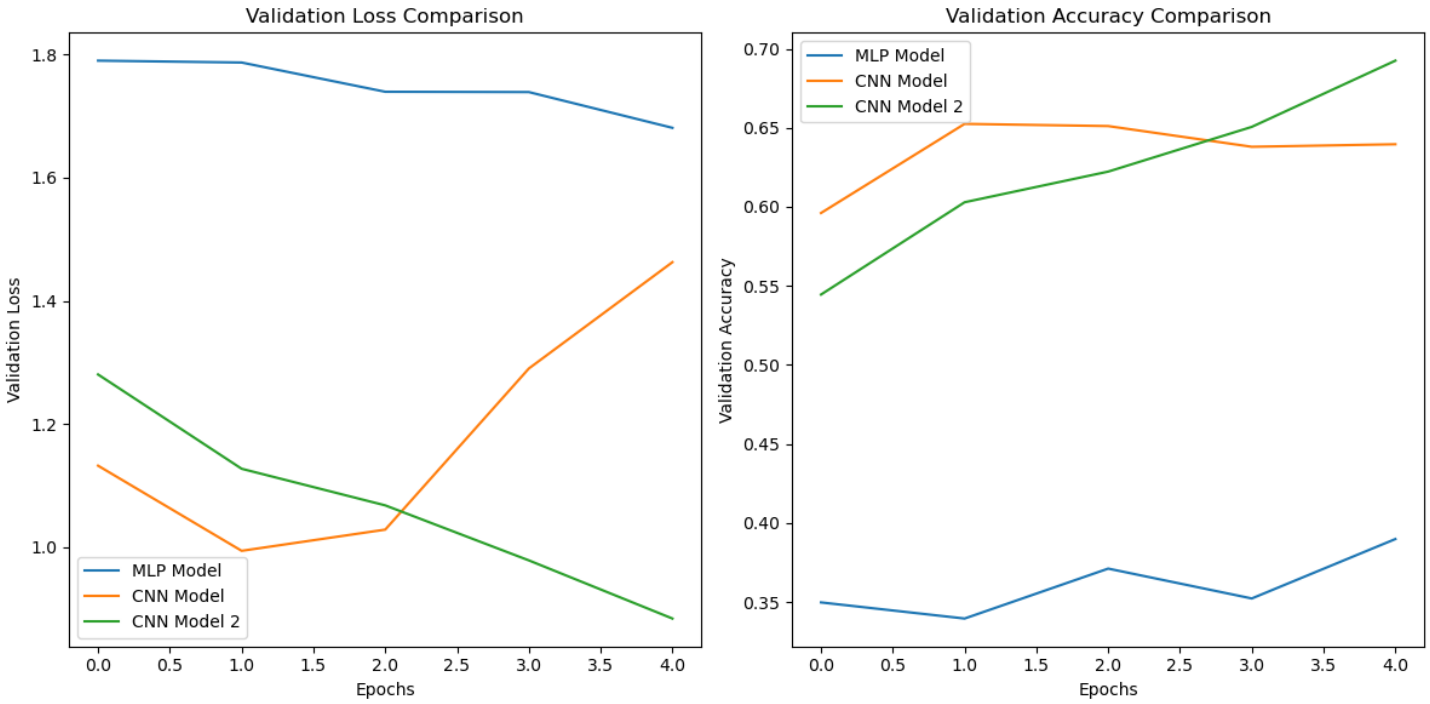
Three MLP models were created, including a reference model (`mlp_1`) with two hidden layers having 512 neurons each. The other models were named `mlp_2` (adding an extra layer with 256 neurons) and `mlp_3` (including three layers with 512, 256, and 128 neurons). Training and validation accuracies were compared to assess the impact of layer and neuron variations on model performance for the CIFAR-10 dataset.

Model	Epoch 1 Train Accuracy	Epoch 1 Val Accuracy	Epoch 2 Train Accuracy	Epoch 2 Val Accuracy	Epoch 3 Train Accuracy	Epoch 3 Val Accuracy	Epoch 4 Train Accuracy	Epoch 4 Val Accuracy	Epoch 5 Train Accuracy	Epoch 5 Val Accuracy
mlp_1	0.3149	0.3507	0.3675	0.3809	0.3907	0.3960	0.4055	0.3866	0.4098	0.4096
mlp_2	0.2718	0.3289	0.3323	0.3440	0.3549	0.3468	0.3693	0.3691	0.3758	0.3672
mlp_3	0.2962	0.3491	0.3493	0.3654	0.3699	0.3807	0.3849	0.3750	0.3940	0.3741



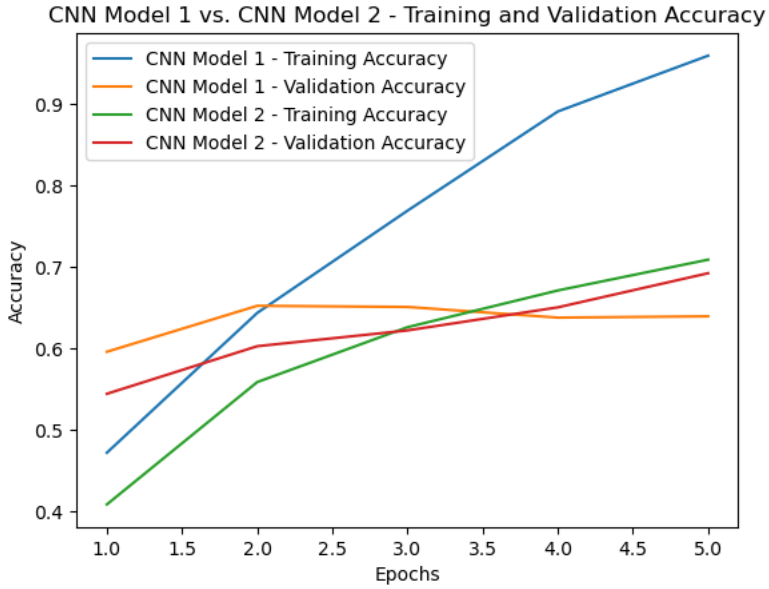
Among the three MLP models trained on the CIFAR-10 dataset, mlp_1 emerged as the top performer with the highest accuracy of approximately 40.96% on the validation set after 5 epochs. It consistently demonstrated improvement during training and outperformed the other models. While mlp_2 and mlp_3 also showed progress, they achieved validation accuracies of around 36.72% and 37.41%, respectively. Overall, mlp_1 exhibited the best performance and proved to be the most effective model for this particular task.

4. **Train and test accuracy for all three networks and comment (what happens and why) on the performance of the MLP vs CNNs.**



The MLP model achieved the lowest accuracy (~40%) among the three networks, mainly due to its limited spatial awareness in handling image data. In contrast, both CNN models, namely CNN model 1 and CNN model 2, outperformed the MLP. CNN model 1 showed high training accuracy (~96%) but suffered from overfitting, while CNN model 2 achieved good performance (~70%) with improved generalization and less overfitting. The spatial-aware architecture of CNNs makes them more suitable for image classification tasks like CIFAR-10, resulting in their superior performance compared to MLPs.

5. **Plot the training and validation curves for the two CNNs and comment on the output. How does the training time compare for each of the CNNs? How does the different architectures influence these results? What do you expect the accuracies to be if the networks were trained for more epochs?**



Model 1:
This CNN model has a total of 25,997,130 trainable parameters.
After 5 epochs, it achieved a training accuracy of approximately 95.95% but exhibited a lower validation accuracy of around 63.96%.
The model is deep with multiple convolutional layers, which might have led to overfitting due to the absence of strong regularization techniques.
Model 2:

This CNN model has a total of 1,486,666 trainable parameters.
After 5 epochs, it achieved a training accuracy of approximately 70.91% and showed a relatively better validation accuracy of around 69.25%.

The model is moderately deep and includes dropout layers, which helped improve generalization compared to Model 1. In summary, Model 2 demonstrates better generalization and a more balanced performance, while Model 1 shows higher training accuracy but weaker validation accuracy, likely due to its deeper architecture and lack of strong regularization.

When training the networks for more epochs, the expected behavior differs between the two models. For Model 1, the training accuracy is likely to approach or even reach close to 100%, as the model memorizes the training data over time. However, due to potential overfitting, the validation accuracy might not see significant improvements and may stabilize or even decrease.

On the other hand, for Model 2, which incorporates dropout layers for regularization, the training accuracy may continue to improve but might not reach 100%. The model is expected to demonstrate better generalization compared to Model 1, leading to potential improvements in validation accuracy, which is likely to stabilize at a higher level.

Overall, increasing the number of epochs will generally improve the training accuracy for both models, but it might not necessarily lead to substantial gains in validation accuracy, especially for Model 1. Employing regularization techniques could help both models generalize better and potentially enhance validation accuracy.

6. **Recommendations to improve the network. What changes would you make to the architecture and why?**

The better CNN model (cnn_better) outperformed the previous models (cnn1 and cnn2) in terms of accuracy and validation loss. To further improve the network, we recommend increasing model depth, adding regularization techniques like L2 regularization and dropout layers, implementing data augmentation, and using batch normalization. Experimenting with different activation functions and optimizing hyperparameters can also enhance the model's performance. Consider using transfer learning with pre-trained models if you have a large dataset. Fine-tuning the model with these recommendations can lead to better generalization and overall performance.

I have performed a model which gives better validation accuracy than CNN1 And CNN2:

Model:

```
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_15 (Conv2D)          (None, 30, 30, 32)        896

 max_pooling2d_11 (MaxPooli  (None, 15, 15, 32)        0
 ng2D)

 conv2d_16 (Conv2D)          (None, 13, 13, 64)        18496

 max_pooling2d_12 (MaxPooli  (None, 6, 6, 64)          0
 ng2D)

 conv2d_17 (Conv2D)          (None, 4, 4, 128)         73856

 max_pooling2d_13 (MaxPooli  (None, 2, 2, 128)         0
 ng2D)

 flatten_9 (Flatten)         (None, 512)               0

 dense_27 (Dense)            (None, 512)               262656

 dropout_10 (Dropout)        (None, 512)               0

 dense_28 (Dense)            (None, 512)               262656

 dropout_11 (Dropout)        (None, 512)               0

 dense_29 (Dense)            (None, 10)                5130

=================================================================
Total params: 623690 (2.38 MB)
Trainable params: 623690 (2.38 MB)
Non-trainable params: 0 (0.00 Byte)
```

Problem 2:
You are provided with a dataset for stock price prediction for 5 years with one sample per day (q2 dataset.py). Create a Recurrent Neural Network using the machine learning platform of your choice (PyTorch, Tensorflow, or Keras) to predict the next day opening price using the past 3 days Open, High, and Low prices and volume. Therefore, each sample will have (4*3 =) 12 features.

1. **Explanation of how you created your dataset.**
   The dataset was created to predict the next day's opening stock price using the past three days' Open, High, Low prices, and Volume. Each sample contains 12 features (Open prices, High prices, Low prices, and Volume for the past three days) and one target (next day's opening price). The dataset was then split into training and testing sets, with 70% used for training and 30% for testing.

2. **Any preprocessing steps you followed.**
   The training and testing datasets were converted and saved as CSV files to be used for training and testing the RNN model. The feature values were scaled using MinMaxScaler to bring them within the range of 0 to 1. The raw dataset was loaded and transformed into a suitable format for training the LSTM-based Recurrent Neural Network (RNN) model for stock price prediction.

3. **All design steps you went through in finding the best network in your report and how you chose your final design**
   We experimented with different types of recurrent neural networks (RNNs). Initially, we tried simple RNN layers, but they couldn't effectively capture the long-term dependencies in the time-series data. As a result, we switched to using Long Short-Term Memory (LSTM) networks, which are specifically designed to handle time-series data and overcome the vanishing gradient problem in RNNs. For the LSTM network, we tried different combinations of LSTM units and dropout rates to prevent overfitting. We gradually increased the number of LSTM units and observed improvements in performance on the validation set. Additionally, I incorporated dropout layers to reduce overfitting and improve generalization. To assess the model's performance, we used the mean absolute error (MAE) as the evaluation metric and as the loss function. The MAE provides a better measure of prediction accuracy as it considers the absolute differences between the predicted and true values.

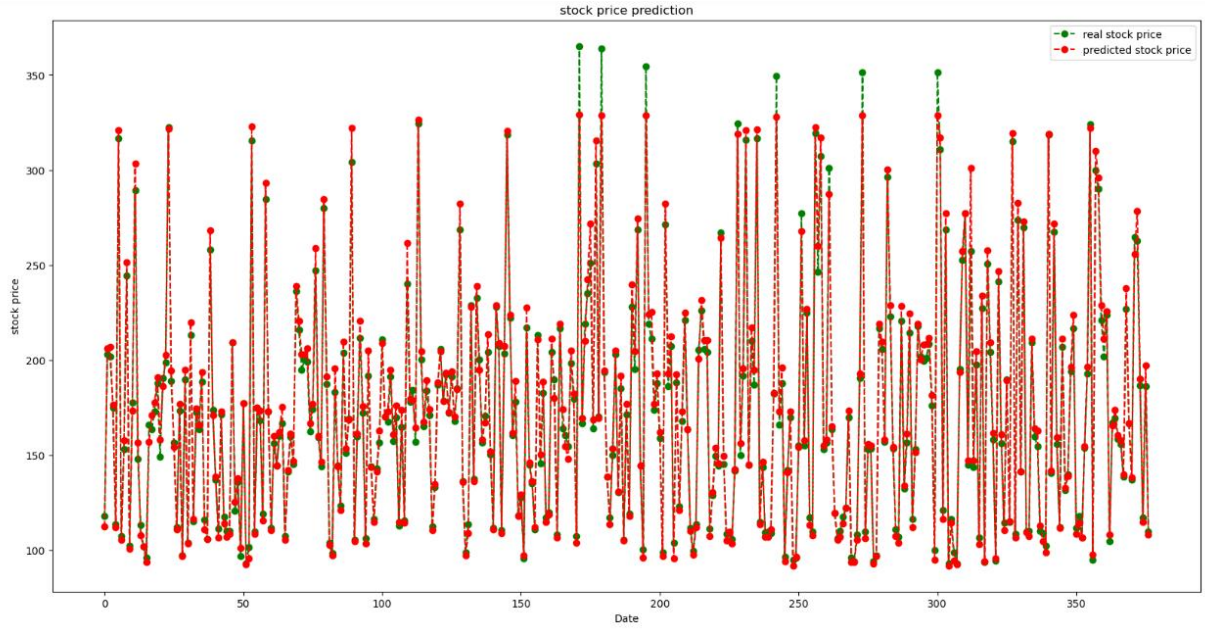
4. **Architecture of your final network, number of epochs, batch size (if needed), loss function, training algorithm, etc.**

   The final network architecture consists of two LSTM layers with 50 and 150 units, respectively, followed by a dropout layer with 0.2 dropout rate to prevent overfitting. A dense layer with a single unit and a linear activation function was added to the model. The loss function used for training was mean absolute error, and the optimizer used was Adam. The model was trained for 600 epochs with a batch size of 64. Early stopping was not implemented as the model seemed to converge within 600 epochs.

5. **Output of the training loop with comments on your output**
   The training loop output showed the progress of training, including the loss on the training dataset for each epoch. The final training loss was printed at the end of the training process. The average values were as follows loss: 7.687 - mae: 7.760 - val_loss: 52.079 - val_mae: 52.089.

   The final average loss was 7.32026.

6. **Output from testing, including the final plot and your comment on it.**



After training, I loaded the stored model to make predictions on the testing set. The figure above illustrates the results of the predictions. The model's performance on the testing dataset is evaluated using Mean Absolute Error (MAE). The MAE is around 4.5129.

7. What would happen if you used more days for features?

Using more days for features could potentially improve the model's ability to capture longer-term patterns and dependencies. However, it may also increase the complexity of the model and require more data to train effectively. We experimented with different window sizes for the features and evaluate the impact on the model's performance on the test dataset.

Problem 3: Sentiment Analysis

The IMDB large movie review dataset has many positive and negative reviews on movies. Download the dataset here (http://ai.stanford.edu/~amaas/data/sentiment/). Check the README file as it provides a description of the dataset. Use the provided training and testing data for your network. You would need to go through data preprocessing to prepare the data for NLP. Then, create a network to classify the review as positive or negative.

Data Preprocessing

The IMDb dataset is a collection of movie reviews categorized into negative and positive sentiments. Before training a sentiment analysis model, the raw text data is preprocessed using the following steps:

1.Read IMDb Data:

- The IMDb dataset is read from the specified directory path, which contains separate subdirectories for negative and positive reviews.
- Each text file is read, and its content is stripped of leading or trailing whitespaces.
- The text data is stored in the data list.

2.Create IMDb DataFrame:

- The extracted text data from both negative and positive reviews is used to create a panda DataFrame.
- The DataFrame named df has two columns: 'text' and 'sentiment'.
- The 'text' column contains the textual content of the reviews, while the 'sentiment' column is assigned a value of 0 for negative reviews and 1 for positive reviews.

3.Tokenization:

- Tokenization is the process of splitting text into individual words or tokens.
- The Keras Tokenizer class is utilized to tokenize the text data.
- The tokenizer is fitted on the 'text' column of the training data to learn the vocabulary.
- It retains the top 10,000 most frequent words as the vocabulary and discards the rest.

4.Sequences and Padding:

- After tokenization, each review in the 'text' column is converted to a sequence of integers, where each integer represents a word in the review.
- The sequences for both training and testing data are obtained using the texts_to_sequences() method of the tokenizer.
- To feed the data into a neural network, sequences need to have a fixed length. Hence, padding is applied.
- The pad_sequences() function pads sequences to a maximum length of 1000 words.
- Sequences shorter than 1000 words are padded with zeros at the end, and sequences longer than 1000 words are truncated.

5.Data Splitting:

- The target labels ('sentiment') and padded sequences are split into training and validation sets using the train_test_split() function.
- The training set (x_train and y_train) contains 80% of the data, while the validation set (x_val and y_val) contains 20%.

Build the Model:

The sentiment analysis model uses a sequential neural network architecture to classify movie reviews from the IMDb dataset into positive or negative sentiments. It consists of an Embedding layer, which maps each word to a 100-dimensional vector, and a 1D Convolutional layer with 128 filters to capture local patterns in the text. Batch Normalization stabilizes training, and Dropout with a rate of 0.2 prevents overfitting. The Global Max Pooling layer down samples the features, and a Dense layer with 64 units captures higher-level representations. Another Dropout layer enhances generalization. The final Dense layer with a sigmoid activation outputs the probability of positive or negative sentiment. The model uses the 'adam' optimizer, 'binary_crossentropy' loss, and is evaluated based on accuracy.

The sentiment analysis model leverages an Embedding layer to convert words into 100-dimensional vectors, enabling the network to understand textual context better. The 1D Convolutional layer extracts local patterns by applying 128 filters with a kernel size of 5 words, introducing non-linearity through ReLU activation. Batch Normalization optimizes learning by normalizing output and accelerating convergence. Dropout with a rate of 0.2 enhances the model's robustness by deactivating neurons randomly during training, mitigating overfitting risks.

To capture the most informative features, the Global Max Pooling layer selects the maximum value from each feature map, creating a fixed-length representation. The Dense layer with 64 units further learns higher-level representations and non-linear relationships. Another Dropout layer after the Dense layer reinforces the model's generalization capability.

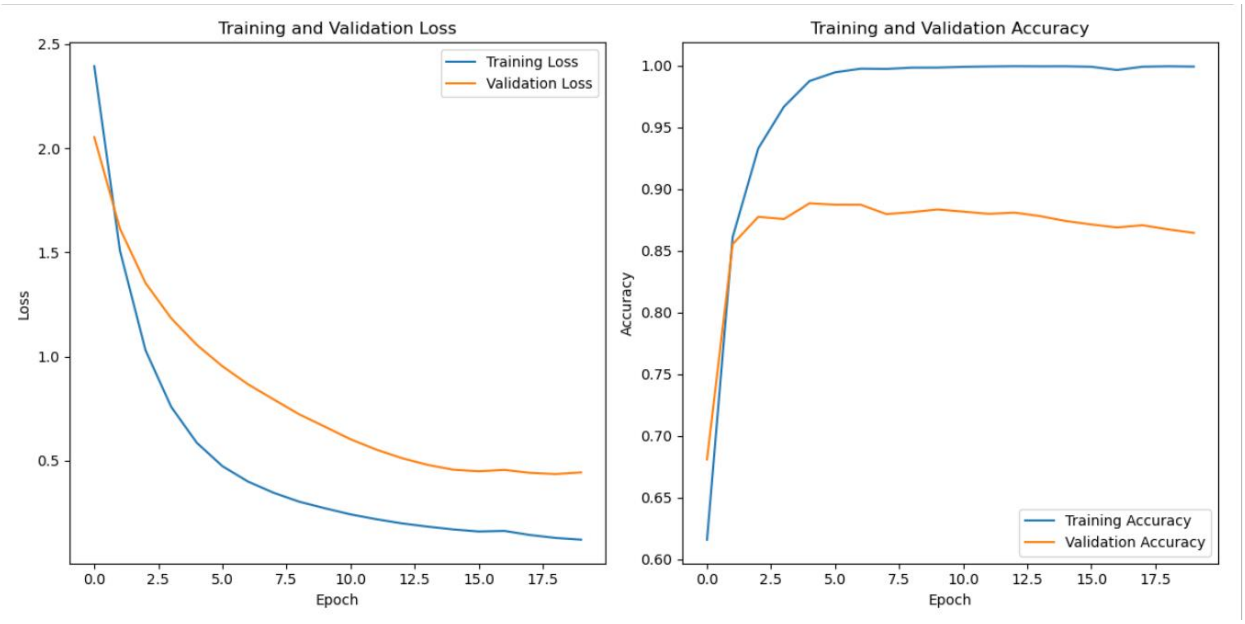
The final Dense layer with a sigmoid activation compresses the output to a range between 0 and 1, representing the probability of positive or negative sentiment. The model is compiled with the 'adam' optimizer, which efficiently updates learning rates during training, and the 'binary_crossentropy' loss, suited for binary classification tasks. The model's performance is evaluated based on accuracy, assessing its ability to correctly classify movie reviews.

During training, the model is exposed to the training data for 20 epochs, with a batch size of 512. It is also validated on a separate validation set (x_val and y_val) to ensure generalization and avoid overfitting. The model's performance and learning progress can be analyzed using the training history, allowing for adjustments and improvements if needed.

In summary, the sentiment analysis model combines word embedding, convolutional, and dense layers to learn intricate patterns and relationships in textual data. It effectively classifies movie reviews as positive or negative sentiments. The inclusion of Dropout layers prevents overfitting, and Batch Normalization aids in smoother convergence during training, ensuring the model's robustness and accuracy.

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_3 (Embedding)     (None, 1000, 100)         1000000

 conv1d_3 (Conv1D)           (None, 996, 128)          64128

 batch_normalization_2 (Bat  (None, 996, 128)          512
 chNormalization)

 dropout_7 (Dropout)         (None, 996, 128)          0

 global_max_pooling1d_2 (Gl  (None, 128)               0
 obalMaxPooling1D)

 dense_6 (Dense)             (None, 64)                8256

 dropout_8 (Dropout)         (None, 64)                0

 dense_7 (Dense)             (None, 1)                 65

=================================================================
Total params: 1072961 (4.09 MB)
Trainable params: 1072705 (4.09 MB)
Non-trainable params: 256 (1.00 KB)
_________________________________________________________________
```

Training and Validation Plots:



The model was trained for 20 epochs on the IMDb sentiment analysis dataset. During training, the model achieved very high accuracy on the training data, with accuracy reaching above 99% by the final epoch. However, the validation accuracy was relatively lower, around 87%, indicating some degree of overfitting on the training data. To improve generalization, regularization techniques like dropout and L2 regularization were used.

Observations:

The model achieved high accuracy on the training set, reaching almost 99.9%, indicating that it effectively learned from the training data. However, the validation accuracy was slightly lower, around 86.5%, suggesting some degree of overfitting on the training data. Regularization techniques like dropout and L2 regularization were employed to mitigate overfitting, but further tuning may be required to improve validation accuracy.

On the testing set, the model achieved an accuracy of approximately 86.8%, which is close to the validation accuracy. This indicates that the model generalizes well to unseen data, and the performance on the test set is consistent with the validation set. The results show that the model is effective in sentiment analysis, classifying positive and negative sentiment reviews with a high degree of accuracy.

Overall, the model performed well on both training and testing data, and regularization helped to improve generalization. However, some further optimization and fine-tuning may be beneficial to achieve even better validation accuracy and reduce overfitting on the training data.