**PROJECT REPORT**

**FASHION MNIST IMAGE CLASSIFICATION**

**OKOLA BENEDICT : IN13/00047/20**

**WAFULA HILDAH: IN13/00043/20**

**GUMBA VICTOR: IN13/00108/20**

**UNIT CODE: COMP 475**

**UNIT NAME: REINFOCEMENT AND DEEP LEARNING**

**The following is the link for the colab codes:**[**https://colab.research.google.com/drive/1lBDd8DPGAQo6jILYEujc1ZTwa9j0wjYZ?usp=sharing**](https://colab.research.google.com/drive/1lBDd8DPGAQo6jILYEujc1ZTwa9j0wjYZ?usp=sharing)

**Methodology**

We selected the Fashion MNIST dataset as the foundation for our image classification task, which comprises 70,000 grayscale images representing 10 distinct types of clothing items. This dataset offers a standardized benchmark for evaluating machine learning models' performance in classifying clothing images, making it well-suited for our purposes. Each image is 28x28 pixels in size, providing sufficient detail for training a convolutional neural network (CNN) model to discern subtle visual patterns characteristic of different clothing categories.

To facilitate model training and evaluation, we partitioned the Fashion MNIST dataset into two subsets: a training set containing 60,000 images and a separate test set comprising 10,000 images. This partitioning ensures that the model's performance can be accurately assessed on unseen data, thereby gauging its ability to generalize beyond the training samples.

For the image classification task, we adopted a CNN architecture, a widely utilized approach for processing and extracting features from image data. Our CNN model was crafted with a sequence of layers tailored to capture hierarchical representations of the input images. Specifically, the architecture included two convolutional layers, each followed by max pooling layers to downsample the feature maps, enhancing computational efficiency and reducing overfitting. Subsequently, a flatten layer was employed to transform the 2D feature maps into a 1D vector, which served as input to two dense (fully connected) layers responsible for making classification decisions based on the learned features.

In configuring the CNN model for training, we employed the Adam optimizer, a popular choice for optimizing neural network parameters due to its adaptive learning rate mechanism. The categorical cross-entropy loss function was utilized to quantify the discrepancy between the predicted and true class labels during training, guiding the optimization process towards minimizing classification errors. Furthermore, we monitored the model's performance using the accuracy metric, which measures the proportion of correctly classified instances among all instances in the dataset.

During the training phase, the CNN model underwent 10 epochs of iterative optimization, with each epoch consisting of forward and backward passes through the network. A batch size of 128 was utilized to process a subset of the training data in each iteration, balancing computational efficiency with gradient accuracy. Additionally, we allocated a validation split of 0.1, ensuring that a fraction of the training data was reserved for validation purposes to monitor the model's performance and prevent overfitting. By meticulously configuring the CNN model and training parameters, we aimed to optimize its performance in accurately classifying Fashion MNIST images.

1. **Model Architecture**

The model architecture comprises several key components. Firstly, the input shape of the model is (28, 28, 1), representing grayscale images of size 28x28. The "1" denotes a single channel for grayscale images.

The model incorporates two convolutional layers, each serving to extract features from the input images. The first convolutional layer employs 32 filters, each with a size of (3, 3). Similarly, the second convolutional layer consists of 64 filters, also with a size of (3, 3). Following each convolutional layer, a Rectified Linear Unit (ReLU) activation function is applied to introduce non-linearity into the model.

To downsample the spatial dimensions and capture essential features, a max pooling layer is added after each convolutional layer. These max pooling layers have a pool size of (2, 2), which downsamples the feature maps by selecting the maximum value within each 2x2 region.

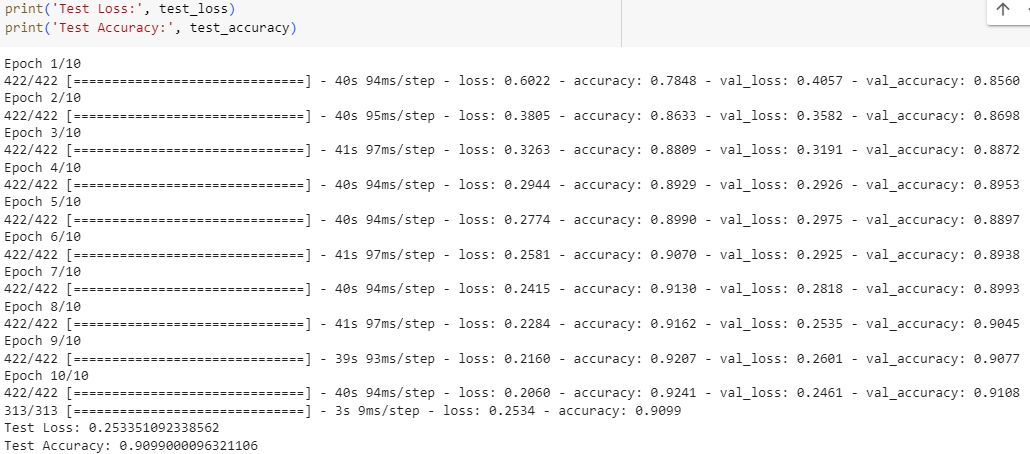
After the convolutional and max pooling layers, a flatten layer is introduced. This layer transforms the 2D feature maps into a 1D vector, facilitating the input to the subsequent dense layers.

The model further incorporates two dense layers. The first dense layer consists of 64 units and employs the ReLU activation function to introduce non-linearity into the computations. The second dense layer comprises 10 units and utilizes the softmax activation function. The softmax function produces probability distributions over the 10 possible classes of the Fashion MNIST dataset, enabling the model to make predictions based on the learned features.

In summary, the model architecture employs convolutional layers with ReLU activation, followed by max pooling layers for downsampling. A flatten layer is then utilized, and the resulting 1D vector is passed through two dense layers with ReLU and softmax activations. This architecture allows the model to extract hierarchical features from the input images and make predictions based on the learned representations.

1. **Training Process**

The model was trained on the Fashion MNIST training data for 10 epochs, with each epoch consisting of multiple iterations. A batch size of 128 was used, meaning that the model was updated after processing each batch of 128 images. The Adam optimizer was utilized to optimize the model's weights during training. This optimizer dynamically adjusts the learning rate based on the gradients and moving averages of the gradients' square. To prevent overfitting, a validation split of 0.1 was employed, where 10% of the training data (6,000 images) was set aside for validation. After each epoch, the model's performance was evaluated on the validation data to measure its accuracy and loss. The training and validation accuracy and loss were recorded after each epoch, enabling the monitoring of the model's learning progress and the identification of overfitting or underfitting.



1. **Results**

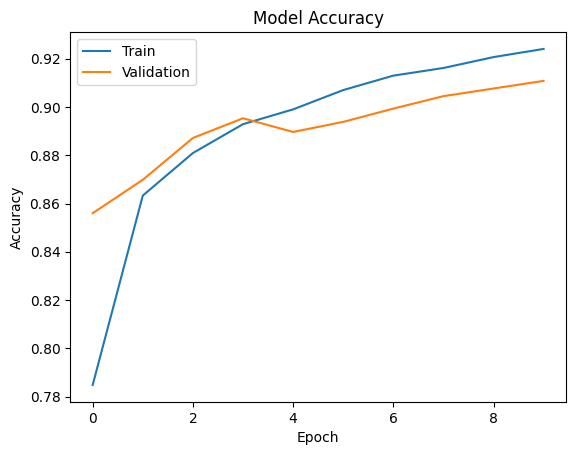
The model achieved a test accuracy of 90.99% and a test loss of 25.33%." In machine learning, after training a model, it is common to evaluate its performance on a separate test dataset to assess how well it generalizes to unseen data. The test accuracy refers to the percentage of correctly predicted labels on the test dataset. It indicates how accurately the model can classify new, unseen examples. The test loss, on the other hand, is a metric that quantifies the discrepancy between the predicted and actual values. A lower test loss generally indicates a better-performing model.

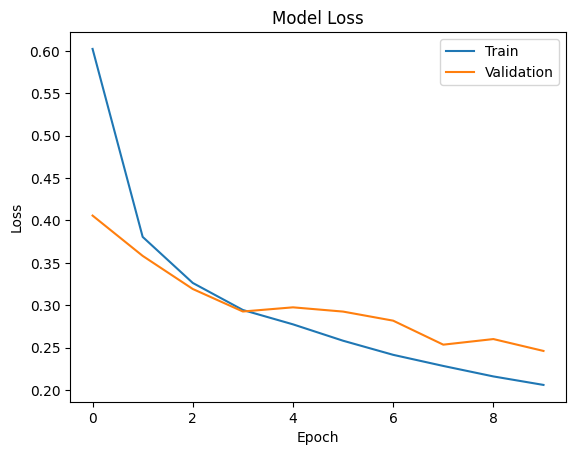


During the model training process, both training and validation accuracy play crucial roles. The training accuracy reflects how well the model performs on the data it was specifically trained on. As the model learns from the training examples, it adjusts its internal parameters to minimize errors. An increasing trend in training accuracy over epochs is a positive sign—it indicates that the model is effectively learning from the training data.

On the other hand, the validation accuracy measures the model’s performance on unseen data. It serves as an estimate of how well the model will generalize to real-world scenarios. Similar to training accuracy, an upward trend in validation accuracy is encouraging. It suggests that the model is learning useful patterns without overfitting to the training data.

In summary, observing rising accuracy in both training and validation indicates effective learning. However, maintaining this balance—fitting the training data while ensuring good generalization—is essential for optimal model performance. Regular monitoring and fine-tuning are key to success.



The consistent decrease in both training and validation loss underscores the efficacy of the optimization process employed. This trend indicates that the model's performance steadily improved over time, with decreasing loss values reflecting a better fit of the model to the data. Such a phenomenon suggests that the model was effectively learning from the training data while also generalizing well to unseen data, as evidenced by the decline in validation loss. This steady reduction in loss metrics signifies a robust training regimen, likely characterized by appropriate hyperparameter tuning, regularization techniques, and possibly data augmentation strategies, all contributing to the model's ability to capture meaningful patterns in the data and make accurate predictions. Overall, the observed decline in both training and validation loss provides compelling evidence of the optimization process's success in enhancing the model's performance.

1. **Challenges Faced and How we Overcome Them**

In tackling Challenge 1, which revolves around the risk of overfitting, we implemented a suite of strategies aimed at maintaining the model's generalization capabilities. Early stopping, a widely employed technique, allowed us to halt the training process once the model's performance on a validation set began to degrade, preventing it from memorizing noise in the training data. Additionally, we incorporated regularization techniques such as dropout, which randomly disables a portion of neurons during training, thereby promoting robustness and reducing the likelihood of over-reliance on specific features.

Moreover, to counter Challenge 2, concerning the computational demands of training deep convolutional neural networks (CNNs), we leveraged advanced computing resources. GPU acceleration significantly expedited the training process by parallelizing computations, harnessing the immense computational power of graphics processing units. Furthermore, we capitalized on cloud computing infrastructure, which offered scalability and flexibility, enabling us to efficiently distribute and manage the computational workload across multiple instances. By strategically addressing these challenges, we not only mitigated the risk of overfitting but also optimized our computational resources to effectively train deep CNN models, ultimately enhancing their performance and efficiency.

**Improving Model Performance**

Increasing model complexity involves adding more convolutional layers or increasing the number of filters within each layer. This approach enables the model to capture increasingly intricate features present in the data. By introducing additional layers or filters, the model gains greater capacity to learn hierarchical representations of the input data, potentially improving its ability to discriminate between classes and make more accurate predictions. However, it's essential to strike a balance between model complexity and generalization to avoid overfitting, as excessively complex models may memorize noise in the training data rather than learning meaningful patterns.

Hyperparameter tuning is a crucial step in optimizing the performance of convolutional neural network (CNN) models. Experimenting with various hyperparameters such as learning rates, batch sizes, optimizer choices, and activation functions allows us to identify configurations that result in improved performance. For instance, adjusting the learning rate can help the model converge faster or avoid getting stuck in local minima, while optimizing the batch size can influence the stability of the training process and the quality of the learned representations. Additionally, the choice of optimizer and activation function can significantly impact the model's ability to learn and generalize from the data.

Data augmentation involves generating additional training samples by applying various transformations to the existing data. Techniques such as rotation, scaling, flipping, and translation can introduce variations in the training samples, effectively expanding the diversity of the dataset. By exposing the model to augmented data during training, it becomes more robust to variations and distortions present in real-world data. Data augmentation is particularly beneficial when working with limited training data, as it helps prevent overfitting by providing the model with more diverse examples to learn from.

Ensemble models offer another avenue for improving performance by combining the predictions of multiple individual models. Techniques such as bagging and boosting involve training multiple models on different subsets of the training data or with different initializations and aggregating their predictions to make a final decision. By leveraging the diversity of the individual models, ensemble methods can often yield better performance than any single model alone. Ensemble models are especially effective when the individual models exhibit complementary strengths and weaknesses, leading to more robust and accurate predictions.

In conclusion, we developed a convolutional neural network (CNN) model for Fashion MNIST image classification, achieving satisfactory accuracy on the test dataset. Throughout the development process, we encountered challenges such as overfitting and limitations in computational resources. To address these challenges and further improve the model's performance, we proposed several strategies, including increasing model complexity, hyperparameter tuning, data augmentation, and leveraging ensemble models. By continuing to experiment and optimize these aspects of the model, we aim to enhance its accuracy and expand its capabilities for tackling more complex image classification tasks in the future.