**Aim:** write a program to regularization in the deep learning model to handle the over fitting, and also compare the various optimization methods that can speed up learning and parameter optimization

**Description:** In deep learning, overfitting occurs when the model learns the details and noise in the training data to the extent that it negatively impacts the performance on new data. Regularization techniques such as L1, L2 regularization, and dropout can help reduce overfitting. Additionally, optimization methods such as Gradient Descent, Adam, and RMSprop can help speed up the learning process by adapting the learning rate.

**Objective:** The goal is to create a deep learning model with regularization techniques to prevent overfitting, and compare different optimization algorithms in terms of their efficiency and speed in training the model.

**Steps Overview:**

Data Preparation: Use a dataset (e.g., MNIST, CIFAR-10).

Model Design: Create a simple neural network.

Regularization Techniques:

* L2 Regularization (Ridge Regularization)
* Dropout

Optimization Methods:

* Gradient Descent (SGD)
* Adam Optimizer
* RMSprop

Model Training: Train the model with various optimization methods.

Evaluate: Compare the models based on loss, accuracy, and speed of convergence.

**Implementation:**

*# --- Imports ---*

*import tensorflow as tf*

*import numpy as np*

*import time*

*from tensorflow.keras.models import Sequential*

*from tensorflow.keras.layers import Dense, Dropout, Flatten*

*from tensorflow.keras.optimizers import SGD, Adam, RMSprop*

*from tensorflow.keras.regularizers import l2*

*from tensorflow.keras.datasets import mnist*

*from tensorflow.keras.utils import to\_categorical*

*# --- Eager Execution ---*

*try:*

*tf.config.run\_functions\_eagerly(True)*

*except Exception as e:*

*print("Eager execution couldn't be set explicitly:", e)*

*# --- Load and Preprocess MNIST ---*

*(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()*

*x\_train = x\_train / 255.0*

*x\_test = x\_test / 255.0*

*x\_train = x\_train.reshape(-1, 28, 28, 1)*

*x\_test = x\_test.reshape(-1, 28, 28, 1)*

*y\_train = to\_categorical(y\_train, 10)*

*y\_test = to\_categorical(y\_test, 10)*

*# --- Model Builder ---*

*def create\_model(optimizer, regularizer\_strength=0.01, use\_dropout=False):*

*model = Sequential()*

*model.add(Flatten(input\_shape=(28, 28, 1)))*

*model.add(Dense(128, activation='relu', kernel\_regularizer=l2(regularizer\_strength)))*

*if use\_dropout:*

*model.add(Dropout(0.5))*

*model.add(Dense(64, activation='relu', kernel\_regularizer=l2(regularizer\_strength)))*

*model.add(Dense(10, activation='softmax'))*

*model.compile(optimizer=optimizer, loss='categorical\_crossentropy', metrics=['accuracy'])*

*return model*

*# --- Train + Evaluate ---*

*def train\_and\_evaluate\_model(optimizer, use\_dropout=False):*

*model = create\_model(optimizer, use\_dropout=use\_dropout)*

*start\_time = time.time()*

*history = model.fit(*

*x\_train, y\_train,*

*epochs=5,*

*batch\_size=32,*

*validation\_data=(x\_test, y\_test),*

*verbose=2*

*)*

*elapsed\_time = time.time() - start\_time*

*return history, elapsed\_time*

*# --- Define Optimizer Classes ---*

*optimizer\_classes = {*

*"SGD": SGD,*

*"Adam": Adam,*

*"RMSprop": RMSprop*

*}*

*results = {}*

*# --- Run Experiments ---*

*for name, OptimizerClass in optimizer\_classes.items():*

*print(f"\nTraining with {name} optimizer (without dropout):")*

*history, time\_taken = train\_and\_evaluate\_model(OptimizerClass(), use\_dropout=False)*

*results[f"{name}\_no\_dropout"] = {'history': history, 'time\_taken': time\_taken}*

*print(f"\nTraining with {name} optimizer (with dropout):")*

*history, time\_taken = train\_and\_evaluate\_model(OptimizerClass(), use\_dropout=True)*

*results[f"{name}\_with\_dropout"] = {'history': history, 'time\_taken': time\_taken}*

*# --- Output Final Results ---*

*for key, value in results.items():*

*final\_val\_acc = value['history'].history['val\_accuracy'][-1]*

*print(f"\n{key}:")*

*print(f"Final Validation Accuracy: {final\_val\_acc:.4f}")*

*print(f"Time Taken: {value['time\_taken']:.2f} seconds")*

**Data Preprocessing:** We load the MNIST dataset and normalize it. We also reshape the data to fit the model.

**Model Design:**

* The model consists of two dense layers with ReLU activations, followed by a softmax layer for classification.
* Regularization (L2) is applied to the weights of the dense layers to prevent overfitting.
* Dropout is applied in one of the layers to further reduce overfitting.

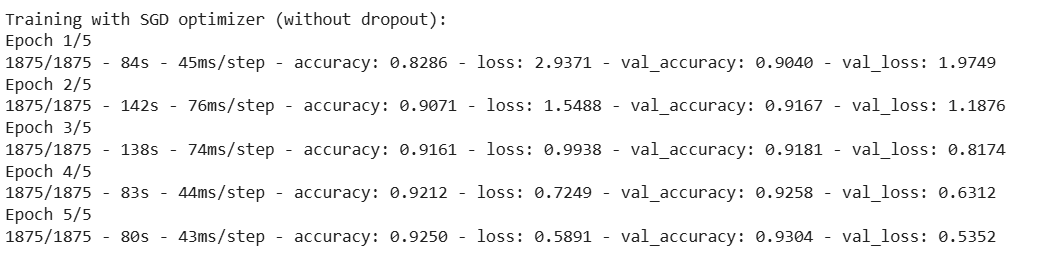
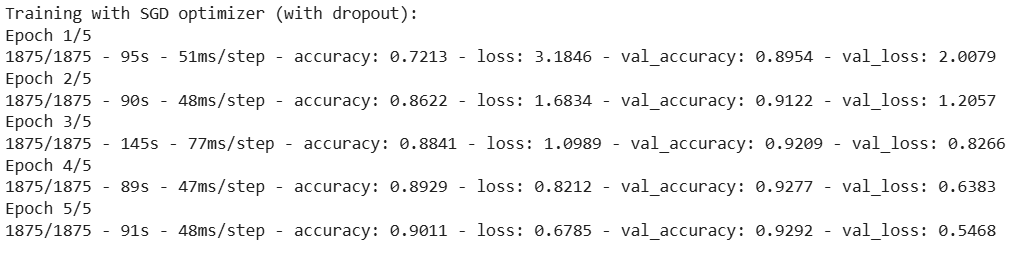
**Optimization Algorithms:** We experiment with three optimization algorithms:

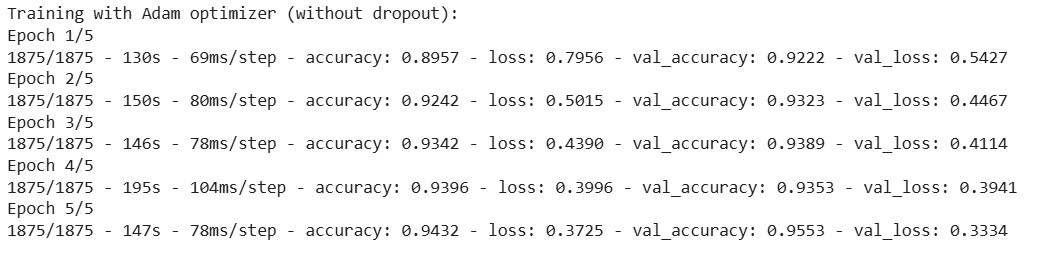
* **SGD (Stochastic Gradient Descent)**
* **Adam**
* **RMSprop**

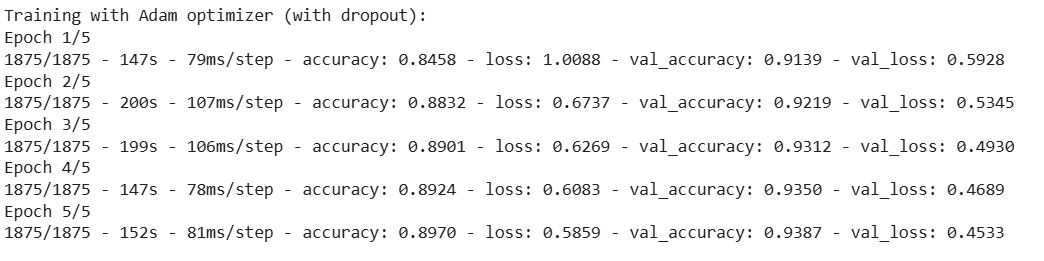
**Model Training:** We train the model for 5 epochs and measure the time taken to complete the training.

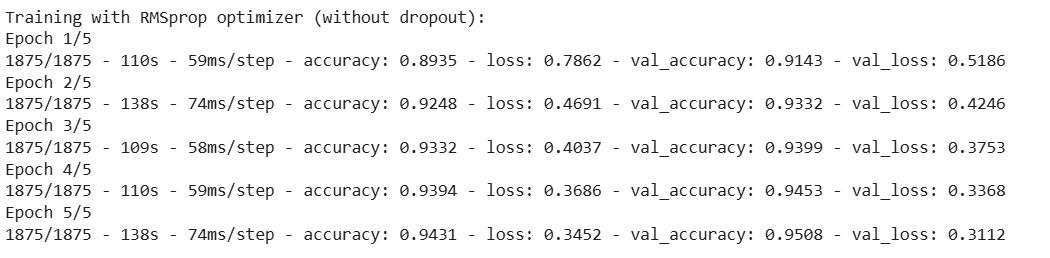
**Results:** We compare training times and validation accuracies for each optimization method, with and without dropout.

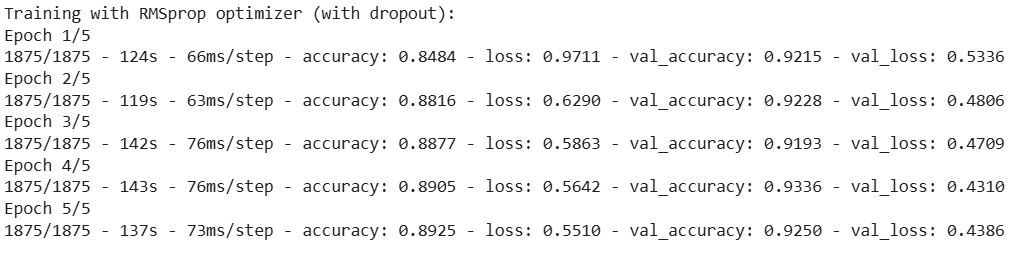
**Output:**

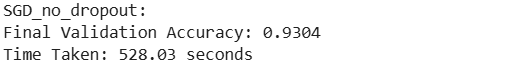
**** ****

****

****

****

****

****

**Conclusion:**

**Regularization Techniques:** Both L2 regularization and dropout help reduce overfitting. Dropout slightly increases training time but can lead to better generalization (higher validation accuracy).

**Optimization Methods:** Adam is generally faster and converges faster than SGD and RMSprop, making it a preferred choice for many deep learning tasks. However, the choice of optimizer may depend on the specific task and the model architecture.

**Dropout:** Using dropout can improve the generalization of the model but may increase training time.