Background

Neural networks, especially in computer vision, excel at complex tasks. They're composed of layers of interconnected nodes (neurons) that learn from input data. In image classification, they automatically extract features from raw pixel values.

Fashion-MNIST dataset has 60,000 training images, 10,000 testing images, in 10 classes. Each is a grayscale 28x28 pixel image representing different fashion items like T-shirts, trousers, dresses, etc.

Experimental Setup:

Fashion-MNIST was accessed via the Keras API, which offers an easy way to load the dataset. Before training, images were scaled to a range of [0, 1] and reshaped to fit the neural network's input dimensions. We considered four key aspects in our experiments:

Learning Rates: We tested a range of learning rates to assess their impact on training dynamics and final accuracy. Learning rates of 0.001, 0.01 and 0.1 were chosen for comparison.

Optimizers: Four different optimizers were investigated: Stochastic Gradient Descent (SGD), Adam, Nadam and RMSprop. Each optimizer was evaluated to determine its influence on convergence and final performance.

Batch Sizes: We experimented with batch sizes of 128 and 256. Varying batch sizes can affect the stability and speed of training, as well as the generalization capabilities of the model.

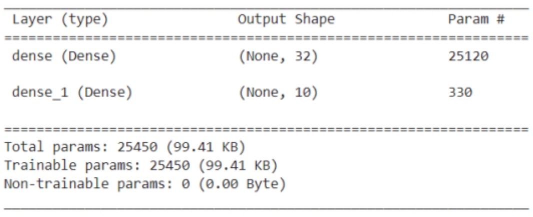
Architectures: We designed and trained for distinct neural network architectures. These varied in terms of depth and complexity, a convolutional neural network (CNN), and CNN with dropout techniques.

Experiments:

Building The Architecture

The architecture consists of two densely connected layers. The input layer has 32 units and uses the sigmoid activation function. The input_shape parameter is specified to match the shape of our training data, which is 784 (28x28) in this case.

The output layer consists of 10 units, representing the ten unique classes in the Fashion-MNIST dataset. The activation function used in the output layer is softmax, which is suitable for multi-class classification tasks.



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 32)                25120

 dense_1 (Dense)             (None, 10)                330

=================================================================
Total params: 25450 (99.41 KB)
Trainable params: 25450 (99.41 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

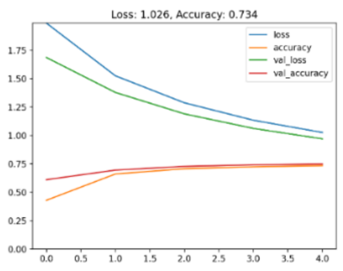
The following will be experimenting with different settings to compile, train the network architecture as defined above with different hyperparameters

Model 1 - Compile and Train the model with the following settings:

Optimizer	LR	Loss	Metrics	Batch_size	epochs	validation_split
sgd	0.01	categorical crossentropy	accuracy	128	5	.1

Evaluating the Model:

Loss	Accuracy	Val Loss	Val Accuracy	Test Loss	Test Accuracy
1.026	0.734	0.970	0.748	0.992	0.729



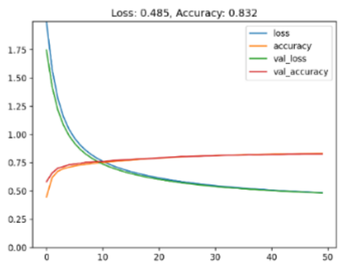
Validation accuracy is slightly higher than training, suggesting no overfitting. Test accuracy being slightly lower is normal as it's unseen data. Loss values align with accuracy. Lower loss is better performance. Next, we'll increase epochs for more learning time. Longer training helps capture complex patterns, but watch for overfitting.

Model 2 - Compile and Train the model with the following settings:

Optimizer	LR	Loss	Metrics	Batch_size	epochs	validation_split
sgd	0.01	categorical crossentropy	accuracy	128	50	.1

Evaluating the Model:

Loss	Accuracy	Val Loss	Val Accuracy	Test Loss	Test Accuracy
0.485	0.832	0.482	0.829	0.509	0.822



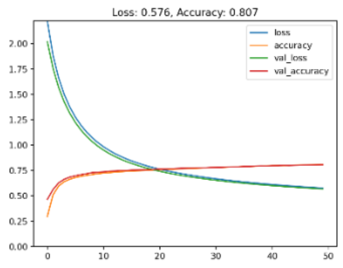
Accuracy improved in all sets (training, validation, test) from the last run, indicating better learning of complex patterns. Lower loss values show improved prediction accuracy. Next, we'll increase the batch size for faster training, processing more samples at once. But, be cautious, very large batches can cause convergence problems.

Model 3 - Compile and Train the model with the following settings:

Optimizer	LR	Loss	Metrics	Batch_size	epochs	validation_split
sgd	0.01	categorical crossentropy	accuracy	256	50	.1

Evaluating the Model:

Loss	Accuracy	Val Loss	Val Accuracy	Test Loss	Test Accuracy
0.576	0.807	0.567	0.806	0.595	0.799



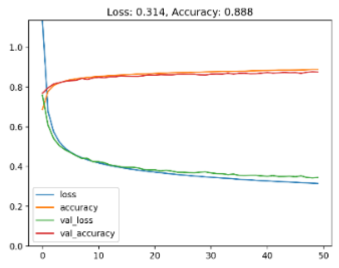
Similar performance to the last run with a batch size of 128. Slight drop in accuracy, small rise in loss, but marginal differences. Increasing batch size didn't yield significant improvements, the model may not benefit much. Next, we'll try a higher learning rate (0.1) for faster convergence, but risk overshooting. It's a trade-off between speed and precision.

Model 4 - Compile and Train the model with the following settings:

Optimizer	LR	Loss	Metrics	Batch_size	epochs	validation_split
sgd	0.1	categorical crossentropy	accuracy	128	50	.1

Evaluating the Model:

Loss	Accuracy	Val Loss	Val Accuracy	Test Loss	Test Accuracy
0.314	0.888	0.344	0.876	0.374	0.867



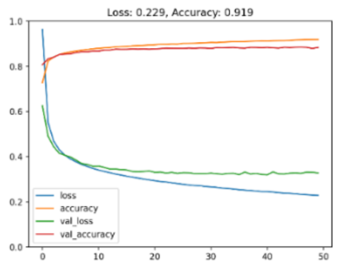
At a learning rate of 0.1, the model shows substantial improvement in training and validation accuracy. Lower loss values indicate better error minimization. In the next model, we experimented with the Adam optimizer.

Model 5 - Compile and Train the model with the following settings:

Optimizer	LR	Loss	Metrics	Batch_size	epochs	validation_split
Adam	0.001	categorical crossentropy	accuracy	128	50	.1

Evaluating the Model:

Loss	Accuracy	Val Loss	Val Accuracy	Test Loss	Test Accuracy
0.229	0.919	0.327	0.883	0.357	0.876



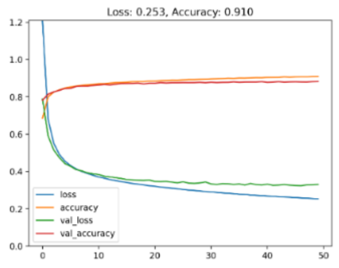
With the Adam optimizer, despite significant improvements in all metrics compared to previous runs, the increasing gap between the training loss and validation loss curves suggests the presence of overfitting. As the number of epochs increases, the disparity between these two curves becomes more pronounced, and there are even instances where the validation loss surpasses the training loss. This indicates that while the model performs well on the training data, its performance on unseen data is declining, which is a possible sign of overfitting. Next, we'll try increasing the batch size.

Model 6 - Compile and Train the model with the following settings:

Optimizer	LR	Loss	Metrics	Batch_size	epochs	validation_split
Adam	0.001	categorical crossentropy	accuracy	256	50	.1

Evaluating the Model:

Loss	Accuracy	Val Loss	Val Accuracy	Test Loss	Test Accuracy
0.253	0.910	0.330	0.883	0.359	0.875



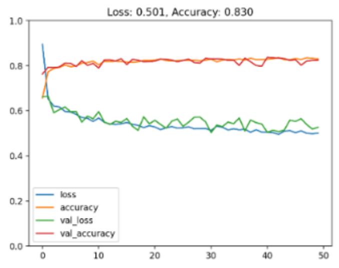
Batch size of 256 yielded similar results to a batch size of 128. Slight drop in accuracy, a small rise in loss, but differences are marginal. Changing batch size didn't bring significant gains; the dataset or model may not benefit much. But overfitting has improved to some extent. Next, we'll raise the learning rate to 0.1 in the next training.

Model 7 - Compile and Train the model with the following settings:

Optimizer	LR	Loss	Metrics	Batch_size	epochs	validation_split
Adam	0.1	categorical crossentropy	accuracy	256	50	.1

Evaluating the Model:

Loss	Accuracy	Val Loss	Val Accuracy	Test Loss	Test Accuracy
0.501	0.830	0.537	0.825	0.557	0.814



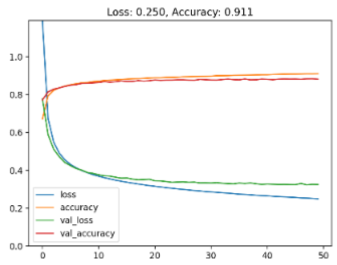
At a learning rate of 0.1, the model's performance changed significantly. Accuracy dropped, loss increased noticeably. This might be too high a learning rate for this dataset given both the loss curve and accuracy curve exhibit non-smooth, fluctuating patterns, causing instability and lower accuracy. Next, we'll switch to the Nadam optimizer for faster convergence.

Model 8 - Compile and Train the model with the following settings:

Optimizer	LR	Loss	Metrics	Batch_size	epochs	validation_split
Nadam	0.001	categorical crossentropy	accuracy	256	50	.1

Evaluating the Model:

Loss	Accuracy	Val Loss	Val Accuracy	Test Loss	Test Accuracy
0.250	0.911	0.325	0.881	0.353	0.876



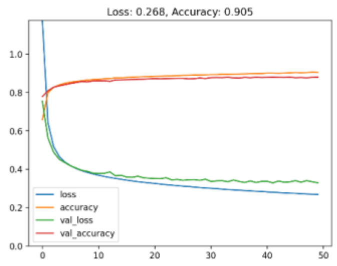
Nadam optimizer led to overall improvement in metrics, however the increasing gap between the training loss and validation loss curves suggests the presence of overfitting. We'll experiment with the rmsprop optimizer in the next model.

Model 9 - Compile and Train the model with the following settings:

Optimizer	LR	Loss	Metrics	Batch_size	epochs	validation_split
rmsprop	0.001	categorical crossentropy	accuracy	256	50	.1

Evaluating the Model:

Loss	Accuracy	Val Loss	Val Accuracy	Test Loss	Test Accuracy
0.268	0.905	0.330	0.880	0.352	0.877



With RMSprop optimizer, the model didn't show significant improvement compared to the Nadam optimiser and it also suggests bit of overfitting given the increasing gap between the training loss and validation loss curves.

At this stage, Model 4 is the best-performing model since its test accuracy is higher than the other models, and it exhibits no overfitting compared to the others. We are considering using Optuna, an automated approach to assess if our current architecture has reached its capacity.

Optuna – Trial

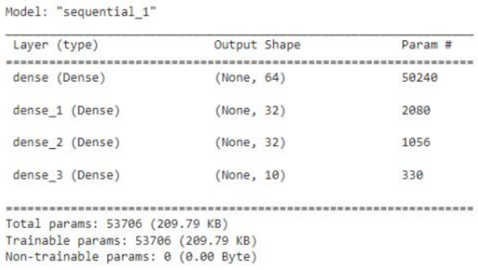
The goal of utilizing Optuna was to ascertain whether our model had reached its capacity.

We use Optuna - Trial to define different learning rates, optimizers, and batch sizes, allowing the model to automatically run and select the best combination. The Optuna best trial achieves the highest test accuracy of 0.858 when tuning the optimizer. However, when we manually tuned the hyperparameters to train our model, we observed that the test accuracy exceeded what was obtained through Optuna (Model 4 has a test accuracy of 0.867).

This discrepancy suggests that manual hyperparameter tuning led to a more effective configuration for our model. Furthermore, we can infer that the network we constructed has likely reached its maximum capacity based on the observed convergence of models to approximately 87% accuracy. This suggests that the current neural network structure is performing optimally given its architecture and the nature of the dataset. Please refer to Appendix 1 for detailed information.

Building a Deeper Architecture

The new architecture utilizes four Dense layers with 64, 32, 32, and 10 neurons respectively, designed for a 10-class task. This deep structure, totaling 53,706 trainable parameters, enhances the model's capacity to learn intricate features and relationships in the data, leading to improved performance.



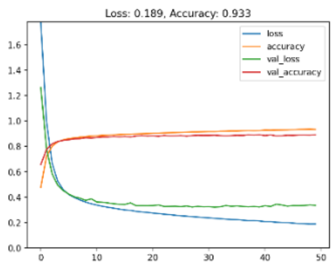
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 64)                50240

 dense_1 (Dense)             (None, 32)                2080

 dense_2 (Dense)             (None, 32)                1056

 dense_3 (Dense)             (None, 10)                330

=================================================================
Total params: 53706 (209.79 KB)
Trainable params: 53706 (209.79 KB)
Non-trainable params: 0 (0.00 Byte)
```

Using the best hyperparameter we obtained from the previous section to train on this deeper architecture.

Compile The Model				**Train The Model**		
Optimizer	**LR**	**Loss**	**Metrics**	**Batch_size**	**epochs**	**validation_split**
Adam	0.001	categorical crossentropy	accuracy	256	50	.1

Evaluating the Model:

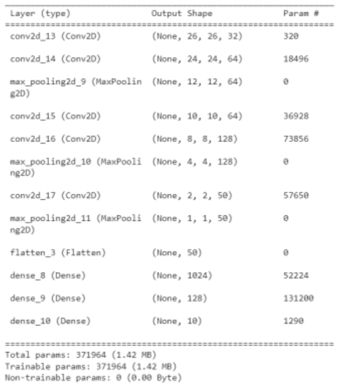
Loss	**Accuracy**	**Val Loss**	**Val Accuracy**	**Test Loss**	**Test Accuracy**
0.189	0.933	0.335	0.891	0.362	0.881



After increasing the complexity, the evaluation metrics show that this model has outperformed Model 4 but it has a higher validation loss which may indicate overfitting. To address this, we will introduce a dropout technique in combination with CNN.

Building a Convolutional Neural Network (CNN) Architecture

CNNs are well-suited for tasks where there are spatial hierarchies of features. For instance, in images, features at higher levels often represent more complex patterns that are built upon simpler patterns at lower levels.

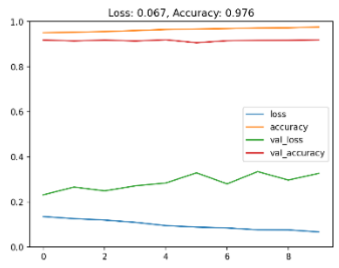


```
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_13 (Conv2D)          (None, 26, 26, 32)        320

 conv2d_14 (Conv2D)          (None, 24, 24, 64)        18496

 max_pooling2d_9 (MaxPoolin  (None, 12, 12, 64)        0
 g2D)

 conv2d_15 (Conv2D)          (None, 10, 10, 64)        36928

 conv2d_16 (Conv2D)          (None, 8, 8, 128)         73856

 max_pooling2d_10 (MaxPooli  (None, 4, 4, 128)         0
 ng2D)

 conv2d_17 (Conv2D)          (None, 2, 2, 50)          57650

 max_pooling2d_11 (MaxPooli  (None, 1, 1, 50)          0
 ng2D)

 flatten_3 (Flatten)         (None, 50)                0

 dense_8 (Dense)             (None, 1024)              52224

 dense_9 (Dense)             (None, 128)               131200

 dense_10 (Dense)            (None, 10)                1290

=================================================================
Total params: 371964 (1.42 MB)
Trainable params: 371964 (1.42 MB)
Non-trainable params: 0 (0.00 Byte)
```

Compile The Model				**Train The Model**		
Optimizer	**LR**	**Loss**	**Metrics**	**Batch_size**	**epochs**	**validation_split**
Adam	0.001	categorical crossentropy	accuracy	256	10	.1

Evaluating the Model:

Loss	**Accuracy**	**Val Loss**	**Val Accuracy**	**Test Loss**	**Test Accuracy**
0.067	0.976	0.327	0.910	0.343	0.916

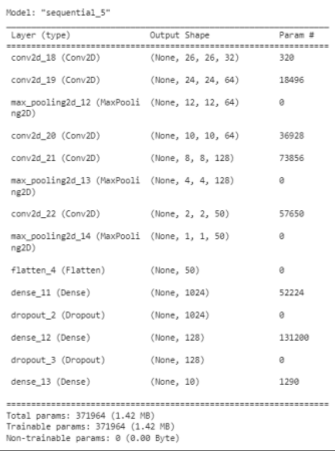


In the model, we have a combination of convolutional layers, max-pooling layers, and dense layers. This architecture allows the model to learn hierarchical features from the input images. Although the model achieves high accuracy on both training and validation sets. There seems to experience overfitting, as the number of epochs increases, the disparity between these two loss curves becomes more pronounced, and the validation loss always surpasses the training loss. This indicates that while the model performs well on the training data, its performance on unseen data is declining, which is a possible sign of overfitting.

In order to maintain the high accuracy and low loss in the model, we will use the dropout technique to reduce overfitting.

Building a Convolutional Neural Network (CNN) Architecture with Dropout Technique

In this architecture, we have added the dropout regularization technique to address the overfitting issue and improve the generalization ability of the model.

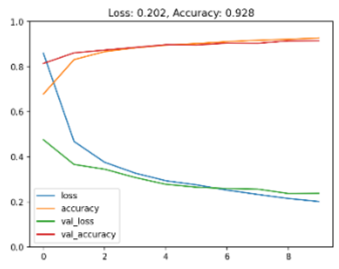


```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_18 (Conv2D)          (None, 26, 26, 32)        320

 conv2d_19 (Conv2D)          (None, 24, 24, 64)        18496

 max_pooling2d_12 (MaxPooli  (None, 12, 12, 64)        0
 ng2D)

 conv2d_20 (Conv2D)          (None, 10, 10, 64)        36928

 conv2d_21 (Conv2D)          (None, 8, 8, 128)         73856

 max_pooling2d_13 (MaxPooli  (None, 4, 4, 128)         0
 ng2D)

 conv2d_22 (Conv2D)          (None, 2, 2, 50)          57650

 max_pooling2d_14 (MaxPooli  (None, 1, 1, 50)          0
 ng2D)

 flatten_4 (Flatten)         (None, 50)                0

 dense_11 (Dense)            (None, 1024)              52224

 dropout_2 (Dropout)         (None, 1024)              0

 dense_12 (Dense)            (None, 128)               131200

 dropout_3 (Dropout)         (None, 128)               0

 dense_13 (Dense)            (None, 10)                1290

=================================================================
Total params: 371964 (1.42 MB)
Trainable params: 371964 (1.42 MB)
Non-trainable params: 0 (0.00 Byte)
```

Compile The Model				**Train The Model**		
Optimizer	**LR**	**Loss**	**Metrics**	**Batch_size**	**epochs**	**validation_split**
Adam	0.001	categorical crossentropy	accuracy	256	10	.1

Evaluating the Model:

Loss	**Accuracy**	**Val Loss**	**Val Accuracy**	**Test Loss**	**Test Accuracy**
0.202	0.928	0.2381	0.915	0.253	0.911



The model performs very well, achieving high accuracy on both training and test sets. The model is performing well on the validation and test sets, indicating that the addition of dropout layers has helped reduce overfitting compared to the previous model.

In conclusion, the final CNN architecture with dropout appears to be a robust and well-performing model for the Fashion-MNIST dataset. Further fine-tuning or experimentation with additional techniques may lead to even better results.

Appendix

				Test Accuracy
Trial 1	**Compile The Model**	**Optimizer**	sgd	0.819(Using Best Trial LR)
		LR	0.00001 to 0.1	
		Loss	categorical crossentropy	
		Metrics	accuracy	
	Train The Model	**Batch_size**	128	
		epochs	50	
		validation_split	.1	
Trial 2	**Compile The Model**	**Optimizer**	sgd	0.853(Using Best Trial Batch size)
		LR	0.01	
		Loss	categorical crossentropy	
		Metrics	accuracy	
	Train The Model	**Batch_size**	32,64,128,256	
		epochs	50	
		validation_split	.1	
Trial 3	**Compile The Model**	**Optimizer**	Adam, Nadam, sgd, rmsprop	0.858(Using Best Trial Optimiser)
		LR	0.1	
		Loss	categorical crossentropy	
		Metrics	accuracy	
	Train The Model	**Batch_size**	256	
		epochs	50	
		validation_split	.1	

Appendix 1 : Optuna – each trial result.