



UNIVERSITÀ DEGLI STUDI  
DI MILANO

---

## Statistical Learning, Deep Learning, and Artificial Intelligence

Image classification using Convolutional Neural Network

With Fashion-MNIST dataset

**Name:** *Natasha Amjad*

**Mat°** 939736

## Contents

1. Abstract: .....	2
2. The Fashion-MNIST dataset: .....	2
3. Goal of Analysis: .....	3
4. Key findings: .....	4
5. Analysis: .....	4
Convolutional Neural Network using Adam: .....	4
Convolutional Neural Network using Stochastic Gradient Descent:.....	6
6. Conclusion: .....	8
7. Appendix:.....	9
(R code) .....	9
8. References:.....	16

## 1. Abstract:

There are various techniques to solve the image classification problem. This project aims to build a simple Machine learning algorithm to unravel the image classification tasks. For this purpose, we'll implement Convolutional Neural Network using TensorFlow which classifies the Fashion-MNIST dataset. it's a dataset of Zalando's article images. We'll see the comparison over Adam and SGD optimizers using convolutional neural network (CNN) within the Fashion-MNIST dataset.

## 2. The Fashion-MNIST dataset:

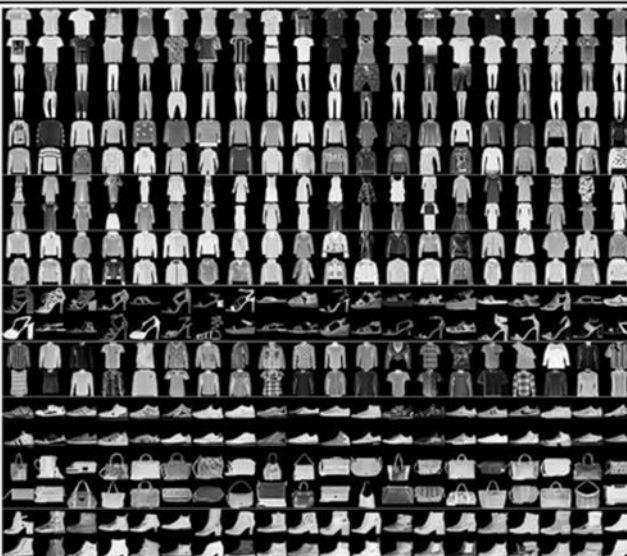
Image classification is one of the most important problems in computer vision, it is very challenging for a computer algorithm to solve it with human-level accuracy.

The fashion-MNIST dataset was proposed as a more challenging substitution of MNIST. Fashion-MNIST is a dataset taken from Zalando's fashion website. Every fashion product on Zalando has different pictures, demonstrating different features of the products, i.e., front and back looks. In our study, we use the front look images of 70, 000 exclusive products to create a Fashion-MNIST dataset. These products are from different gender groups: men, women, kids, and neutral images.

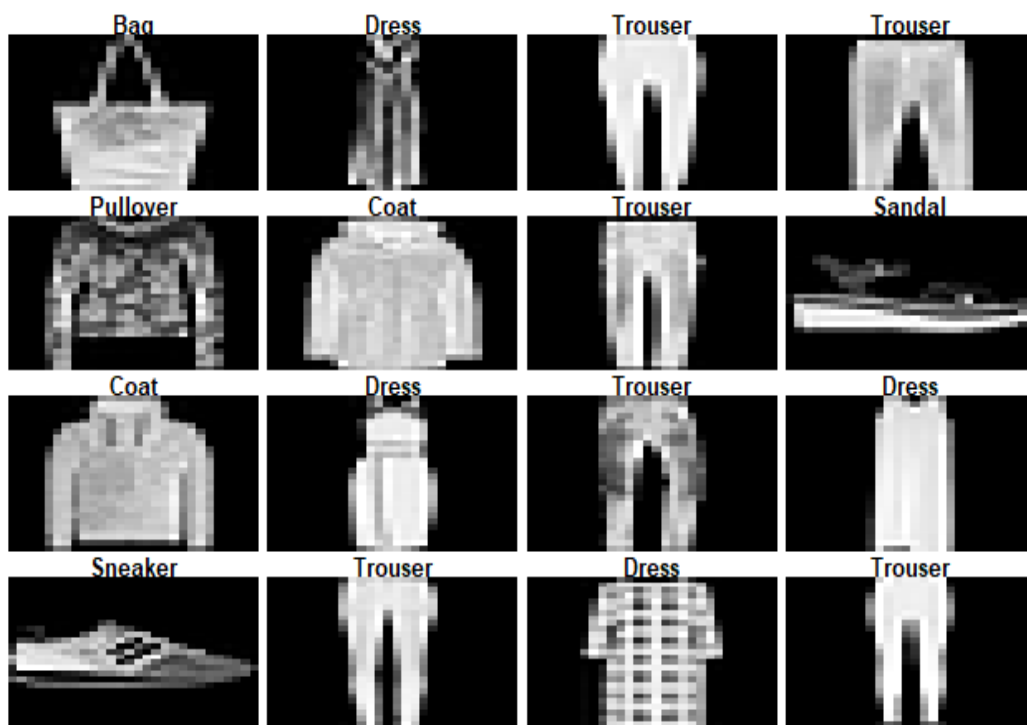
The background of the original picture is light-Gray, and it is stored in 762 × 1000 JPEG format.

For effectively providing different frontend components, we resampled the original picture with multiple resolutions, e.g., large, medium, small, thumbnail, and tiny.

- Our dataset having a training set of 60,000 images and a test set of 10,000 images.
- From each class, 6,000 images are randomly selected.
- Each image is a 28x28 grayscale.
- There are 10 items/classes in these images, which are given below:

Label	Description	Examples
0	T-Shirt/Top	
1	Trouser	
2	Pullover	
3	Dress	
4	Coat	
5	Sandals	
6	Shirt	
7	Sneaker	
8	Bag	
9	Ankle boots	

- Our data is a 3 D-array (image, width, height) of Grayscale values. We prepare our training dataset by transforming 3 D-array into matrices by reshaping the width and height into one single dimension.
- 28x28 image normalize into vectors of length 784 and then converted the Grayscale value from integers ranging 0-255 into floating points ranging 0-1
- The Y data is a vector having a value ranging from 0-9 and to prepare this dataset for training we use one-hot encoding to convert the vectors to binary class matrices.
- We try to plot random images from the training dataset which are given below:



### 3. The goal of Analysis:

The goal of this project is to compare the performance of Adaptive moment estimation (Adam) and Stochastic gradient descent (SGD) using Convolutional Neural Network (CNN) within the Fashion-MNIST dataset.

## 4. Key findings:

- CNN is best for image classification tasks, but it takes an exponential amount of time for computation.
- Adam works well with the default hyperparameter, this is because of its adaptive learning rate property.
- Adam converges more faster while SGD has a convergence problem.
- Adam gives higher accuracy than SGD, and it is consistent across the training and validation set.

## 5. Analysis:

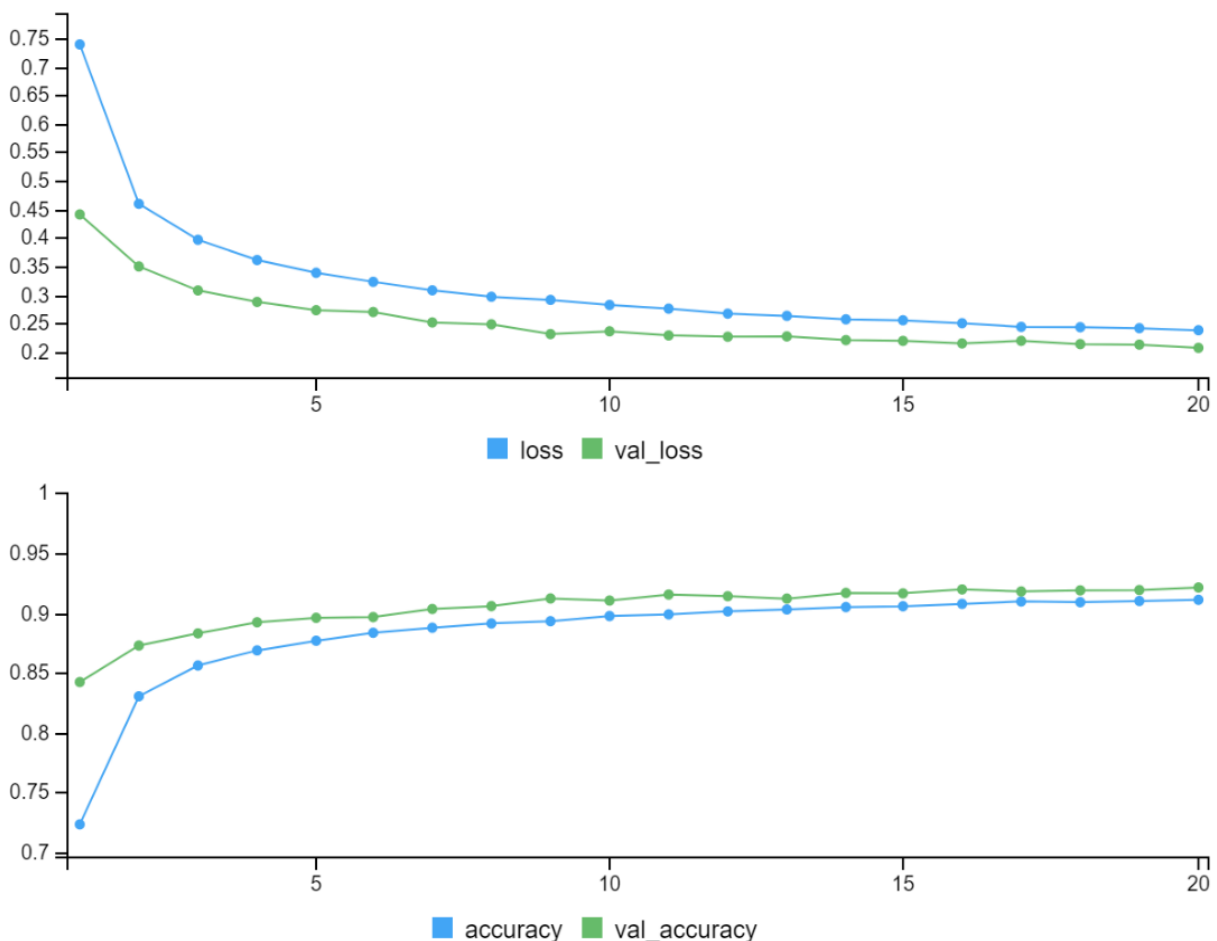
### Convolutional Neural Network using Adam:

Adam is one of the most popular optimizers when we come to CNN. It is a stochastic gradient optimization method that calculates the adaptable learning rates for each parameter. In our analysis, we trained the model with CNN architecture.

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 24, 24, 32)	832
max_pooling2d_1 (MaxPooling2D)	(None, 12, 12, 32)	0
dropout_3 (Dropout)	(None, 12, 12, 32)	0
conv2d_1 (Conv2D)	(None, 10, 10, 64)	18496
max_pooling2d (MaxPooling2D)	(None, 5, 5, 64)	0
dropout_2 (Dropout)	(None, 5, 5, 64)	0
conv2d (Conv2D)	(None, 3, 3, 128)	73856
dropout_1 (Dropout)	(None, 3, 3, 128)	0
flatten (Flatten)	(None, 1152)	0
dense_1 (Dense)	(None, 128)	147584
dropout (Dropout)	(None, 128)	0
dense (Dense)	(None, 10)	1290
Total params: 242,058		
Trainable params: 242,058		
Non-trainable params: 0		

It builds up with a total of 12 layers which include 3 convolutional layers, 4 dropouts, 2 max-pooling, 2 dense, and 1 flatten layer. The dropout rate is used as regularization to decrease the overfitting problem in a network containing many layers, which is set to (0.25, 0.25, 0.3, 0.4) for each different layer. Here in the conv2d layers, we are using 32, 64, 128 filters, and the kernel which specifies the size of the filter in pixels is set to 5x5 in the first conv2d layer to learn larger features and quickly reduce to 3x3 for the next two layers.

Each max-pooling layer is followed by the conv2d for dimension reduction of output volume. The typical non-saturated **ReLU** activation function is used in hidden layers, which allow faster training through relieving the vanishing gradient problem. For the output layer, we applied the **SoftMax** activation function. Which is used for multiclassification as it creates an exponential probability distribution in the range of (0,1) making it easy for classification based on the activation with the highest probability. We use categorical cross-entropy loss function which is well suited in a multi-classification task. It creates the one-hot encoding vector for all categories within the dataset. Each one-hot vector considers as the probability distribution, However, the main demand for this loss function is to compare the two-probability distribution. The model uses it to learn to give the higher probability to correct target image and low to others. We trained the model using Adam optimizer, set the batch size 256 with 20 epochs. We try to evaluate the model accuracy by setting different learning rates, the minimum value set as 0.00001 and the other one as default.



The graph shows the trends of loss and accuracy using Adam with default learning rate it gives the accuracy of 92% And 20% loss while setting LR=0.00001 results in unstable training process but at the end give same accuracy but smaller loss than default one.

## Convolutional Neural Network using Stochastic Gradient Descent:

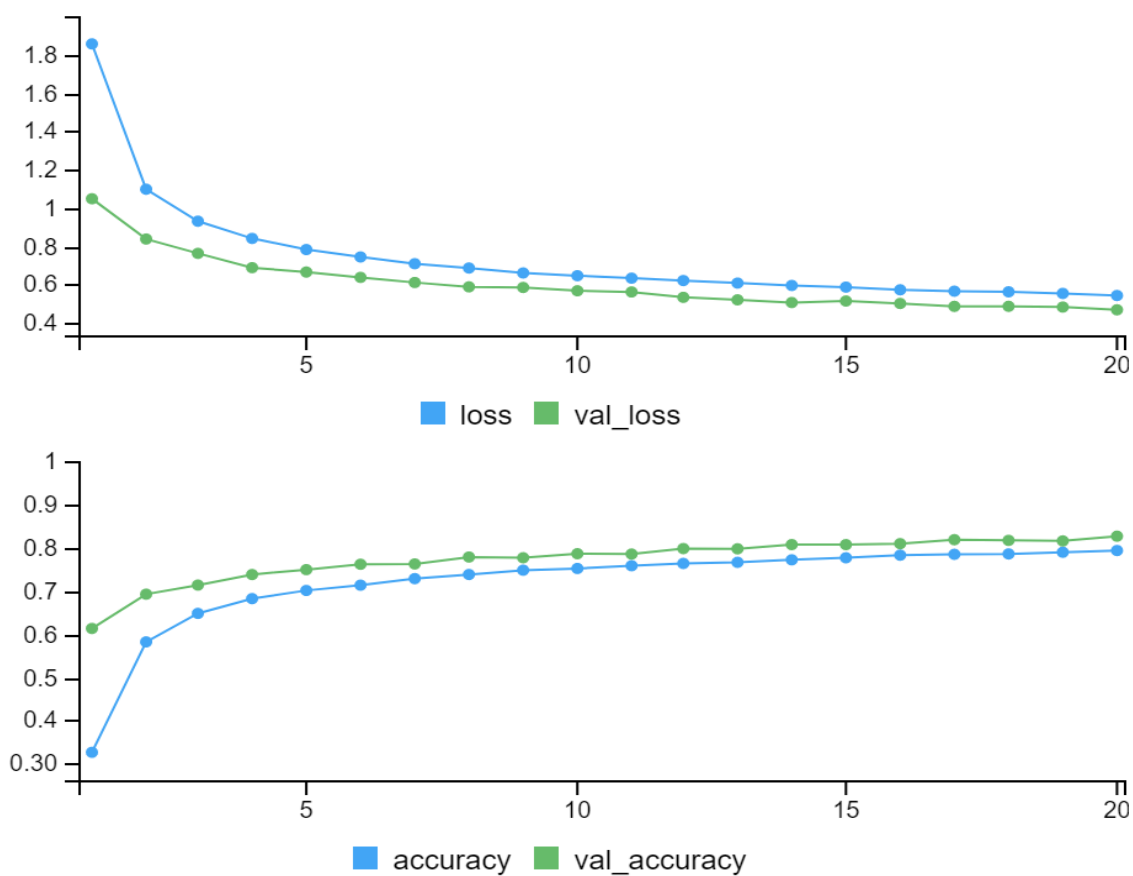
SGD is one of the commonly used optimizers. It maintains a single learning rate throughout the network learning process. In this model, we use a less complex network than the previous one. In total it consists of 11 layers while the kernel size of conv2d is set to 3x3 instead of 5x5 for each layer. There is only one max-pooling layer with a size of 2x2 which removes the dimensionality of the image by reducing the number of pixels in output from the previous convolutional layer.

Layer (type)	Output Shape	Param #
conv2d_35 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d_21 (MaxPooling2D)	(None, 13, 13, 32)	0
dropout_47 (Dropout)	(None, 13, 13, 32)	0
conv2d_34 (Conv2D)	(None, 11, 11, 64)	18496
dropout_46 (Dropout)	(None, 11, 11, 64)	0
conv2d_33 (Conv2D)	(None, 9, 9, 128)	73856
dropout_45 (Dropout)	(None, 9, 9, 128)	0
flatten_11 (Flatten)	(None, 10368)	0
dense_24 (Dense)	(None, 128)	1327232
dropout_44 (Dropout)	(None, 128)	0
dense_23 (Dense)	(None, 10)	1290
Total params: 1,421,194		
Trainable params: 1,421,194		
Non-trainable params: 0		

In this model 4 dropout layers are built, and its main purpose is to slow down the learning curve because this function randomly drops the connections between some neurons during the training phase. Even if a slowdown in the learning rate is less obvious it is still noticeable. The SoftMax activation functions are used in output layers and ReLU in others.

We trained the model using 20 epochs and a Batch size of 256. By fitting the model, we attain 82% validation accuracy.

We also try to set the learning rate for SGD= 0.00001 but the model results in an unstable training process. The figure shows the accuracy and loss trends by setting the default learning rate.



We also observe that the validation performance is higher than training performance, the reason is that using dropout layers in the training process some neurons are randomly deactivated whereas during validation all the units are available so the network has its full computational power and thus it might perform better than in training.

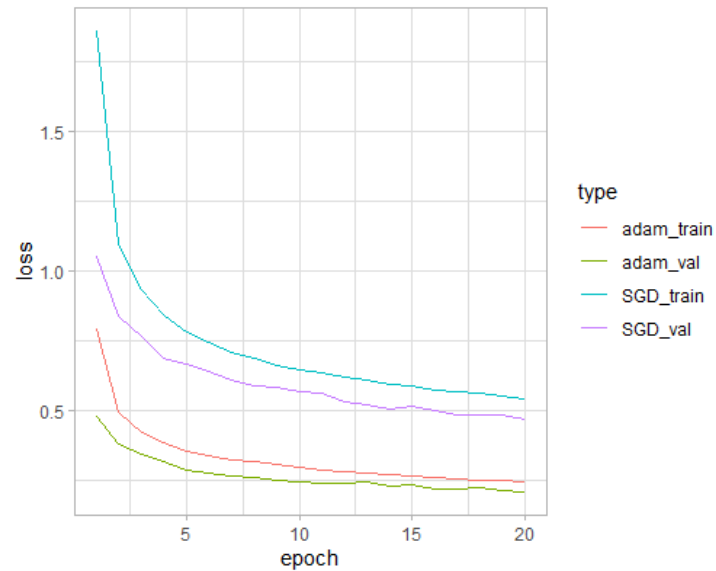


## 6. Conclusion:

In this project, the CNN algorithm has been implemented for image recognition using two optimizers, Adam and SGD. Overall, the model which performed best is with Adam optimizer. It converges more faster than SGD and, its loss is consistently less. However, high accuracy with SGD can be achieved by tuning hyperparameters. To consider a model as best, it must have performed well on both training and validation datasets.

Our analysis showed that using a smaller dataset with augmentation gives good results. To find the best combination of optimizers, activation functions, loss function, learning rate, and other parameters involves a lot of training and testing. In our case, the combination of ReLU with Adam optimizer gave the best performance. Similarly, by using the large batch size with few epochs we attained better accuracy.

While the preference of CNN is essential for image classification, it is clear out that by fixing the value of learning rate and the results attained by different optimizers are substantially variant. It has been seen that a lower LR value permits for a smoother convergence, but it needs more time for a default LR value.



## 7. Appendix:

### (R code)

#### *#Required packages and Libraries for training model*

```
library(knitr)
library(tensorflow)
library(ggplot2)
library(tidyr)
library(tidyverse)
library(readr)
library(keras)
```

#### *# Download the Dataset*

```
Train_Data <- read_csv("fashion-mnist_train.csv",
                      col_types = cols(.default = "i"))
Test_Data <- read_csv("fashion-mnist_test.csv",
                     col_types = cols(.default = "i"))
```

#### *# Fashion MNIST Image Data is 28\*28 pixels*

```
ImgRows <- 28
ImgCols <- 28
```

#### *# Data Preparation*

```
Test_X <- as.matrix(Test_Data[, 2:dim(Test_Data)[2]])
Test_Y <- as.matrix(Test_Data[, 1])
Train_X <- as.matrix(Train_Data[, 2:dim(Train_Data)[2]])
Train_Y <- as.matrix(Train_Data[, 1])
```

#### *# Unflattening the data.*

```
dim(Test_X) <- c(nrow(Test_X), ImgRows, ImgCols, 1)
dim(Train_X) <- c(nrow(Train_X), ImgRows, ImgCols, 1)
```

#### *#Assigning labels to the dataset*

```
Fashion_Labels<-c("T-shirt/top", "Trouser", "Pullover", "Dress", "Coat",
```

```
"Sandal", "Shirt", "Sneaker", "Bag", "Ankle boot")
```

### ***# Function to rotate matrices***

```
rotate <- function(x) t(apply(x, 2, rev))
```

### ***# create function to plot image from a matrix x***

```
plot_image <- function(x, title = "", title.color = "black") {
```

```
  dim(x) <- c(ImgRows, ImgCols)
```

```
  image(rotate(rotate(x)), axes = FALSE,
```

```
        col = grey(seq(0, 1, length = 256)),
```

```
        main = list(title, col = title.color))
```

```
}
```

### ***# Plot images from the training set***

```
par(mfrow=c(4, 4), mar=c(0, 0.2, 1, 0.2))
```

```
for (i in 1:16) {
```

```
  n_row <- i * 10
```

```
  plot_image(Train_X[n_row,,, 1],
```

```
             Fashion_Labels[as.numeric(Train_Data[n_row, 1] + 1)])
```

```
}
```

### ***#Scaling X***

```
Train_X <- Train_X / 255
```

```
Test_X <- Test_X / 255
```

```
#Apply one hot encoding
```

```
Train_Y <- to_categorical(Train_Y, 10)
```

```
Test_Y <- to_categorical(Test_Y, 10)
```

### ***# Hyperparameters Setting***

```
batch_size <- 512
```

```
epochs <- 20
```

```
input_shape <- c(ImgRows, ImgCols, 1)
```

### ***#Convolutional Neural Network Model using sgd.....***

```
model <- keras_model_sequential()

model %>%

  layer_conv_2d(filter = 32, kernel_size = c(3,3), input_shape = c(28, 28, 1), activation =
"relu") %>%

  layer_max_pooling_2d(pool_size = c(2,2)) %>%

  layer_dropout(0.25) %>%

  layer_conv_2d(filter = 64, kernel_size = c(3,3), activation = "relu") %>%

  layer_dropout(0.25) %>%

  layer_conv_2d(filter = 128, kernel_size = c(3,3), activation = "relu") %>%

  layer_dropout(0.4) %>%

  layer_flatten() %>%

  layer_dense(units = 128, activation = "relu") %>%

  layer_dropout(0.4) %>%

  layer_dense(units = 10, activation = "softmax")

summary(model)
```

### ***# Compile the model***

```
model %>% compile(

  loss = loss_categorical_crossentropy,

  optimizer = optimizer_sgd(),

  metrics = c('accuracy')

)
```

### ***# Fit the model***

```
history = model %>% fit(

  Train_X, Train_Y,

  batch_size = batch_size,

  epochs = epochs,

  verbose = 1,

  validation_data = list(Test_X, Test_Y)

)
```

### *#Model Evaluation*

```
History = model %>% evaluate(Test_X, Test_Y, batch_size = 512, verbose = 1)
```

### *#Applying different learning rate than default*

#### *# compile model*

```
model_2 %>% compile(  
  loss = loss_categorical_crossentropy,  
  optimizer = optimizer_sdg(0.00001),  
  metrics = c('accuracy')  
)
```

#### *# fit the model*

```
history_2 = model %>% fit(  
  Train_X, Train_Y,  
  batch_size = batch_size,  
  epochs = epochs,  
  verbose = 1,  
  validation_data = list(Test_X, Test_Y)  
)
```

### *#Convolutional Neural Network using Adam.....*

#### *#Hyperparameters*

```
batch_size <- 512  
num_classes <- 10  
epochs <- 20
```

#### *#Build the model*

```
model_adam <- keras_model_sequential()  
model_adam %>%  
  layer_conv_2d(filters = 32, kernel_size = c(5,5), activation = 'relu',  
    input_shape = input_shape) %>%  
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
```

```
layer_dropout(rate = 0.25) %>%
layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = 'relu') %>%
layer_max_pooling_2d(pool_size = c(2, 2)) %>%
layer_dropout(rate = 0.25) %>%
layer_conv_2d(filters = 128, kernel_size = c(3,3), activation = 'relu') %>%
layer_dropout(rate = 0.4) %>%
layer_flatten() %>%
layer_dense(units = 128, activation = 'relu') %>%
layer_dropout(rate = 0.3) %>%
layer_dense(units = num_classes, activation = 'softmax')
```

```
summary(model_adam)
```

#### ***# Compile the model***

```
model_adam %>% compile(
  loss = loss_categorical_crossentropy,
  optimizer = optimizer_adam(),
  metrics = c('accuracy')
)
```

#### ***# Fit the model***

```
history_adam = model_adam %>% fit(
  Train_X, Train_Y,
  batch_size = batch_size,
  epochs = epochs,
  verbose = 1,
  validation_data = list(Test_X, Test_Y)
)
```

#### ***# Evaluate the model***

```
History = model %>% evaluate(Test_X, Test_Y, batch_size = 512, verbose = 1)
```

### ***#Applying different LR than default***

```
#Compile model
model_adam_2 %>% compile(
  loss = loss_categorical_crossentropy,
  optimizer = optimizer_adam(0.00001),
  metrics = c('accuracy')
)
```

### ***#Fit the model***

```
history_adam_2 = model_adam %>% fit(
  Train_X, Train_Y,
  batch_size = batch_size,
  epochs = epochs,
  verbose = 1,
  validation_data = list(Test_X, Test_Y)
)
```

### ***#Comparison of SGD and Adam***

```
compare <- data.frame(
  SGD_val = history$metrics$val_loss,
  SGD_train = history$metrics$loss,
  adam_val = history_adam$metrics$val_loss,
  adam_train = history_adam$metrics$loss
) %>%
  rownames_to_column() %>%
  mutate(rowname = as.integer(rowname)) %>%
  gather(key = "type", value = "value", -rowname)
```

### *#Plot the result*

```
ggplot(compare, aes(x = rowname, y = value, color = type)) +  
  geom_line() +  
  xlab("epoch") +  
  ylab("loss")
```

### *# Visualization the model predictions*

```
for (i in 1:32){  
  n_row <- i * 10  
  T_Tensor <- Train_X(n_row, , , 1)  
  dim(T_Tensor) <- c(1, ImgRows, ImgCols, 1)  
  pred <- model_adam %>% predict(T_Tensor)  
  plot_image(Train_X(n_row, , , 1),  
    Fashion_Labels(which.max(pred)),  
    "green")  
}
```



## 8. References:

(<https://pdf.sciencedirectassets.com>)

(Classification\_of\_fashion\_article\_images\_using\_convolutional\_neural\_networks.)

(Comparative Study of First Order Optimizers for)

(Fashion-MNIST: a Novel Image Dataset for)