Convolutional Neural Networks for Image Classification on Fashion-MNIST Dataset

Abstract

Recently, deep learning has been widely used in a wide range of industries. Among the deep neural network families, convolutional neural networks (CNNs) yield the most reliable outcomes when applied to real-world problems. Fashion brands have used CNN in their e-commerce to handle a range of problems, such as apparel recognition, search, and recommendation. A key component of each of these techniques is image categorization. This work discusses the concept of classifying fashion-MNIST photos using convolutional neural networks. This research proposes a new CNN-based architecture to train CNN parameters using the Fashion MNIST dataset. This study uses maxpooling, batch normalization, five convolutional layers, dropout and finally linked layers for classification. Various optimizers, learning rates, batch sizes and 100 epochs are used in the experiments. The outcomes displayed that the accuracy of the findings is influenced by the selection of the activation function, dropout rate and optimizer.

On the other hand, the fashion-MNIST dataset includes 28x28 grayscale pictures of 70,000 fashion items from 10 classifications, with 7,000 images in each class. The training set has 48,000 images, the evaluation set contains 6,000 images, and the test set contains 6,000 images. According to experimental results, the suggested model's accuracy exceeded 93%.

1- Proposed CNNs

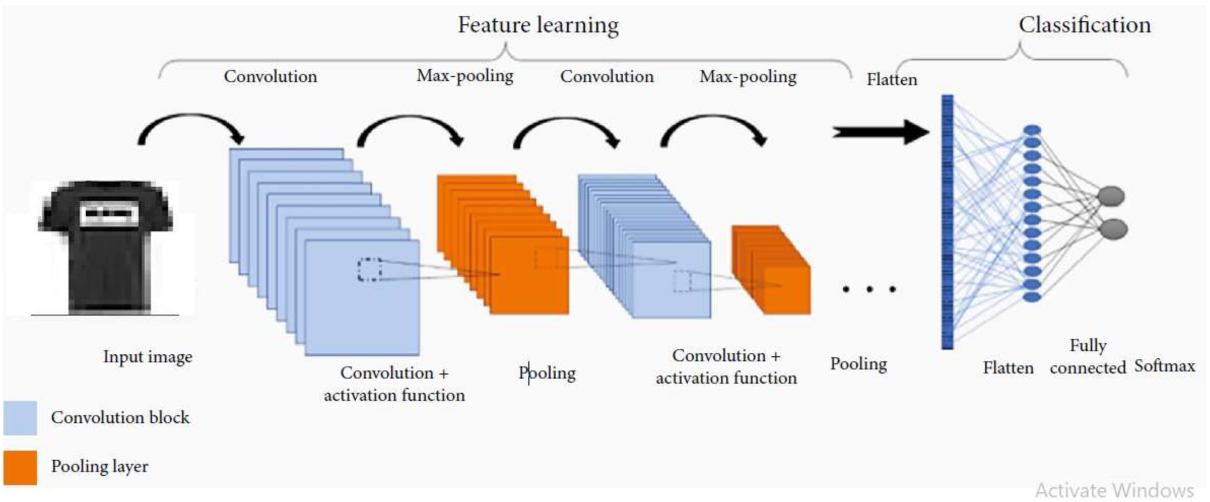
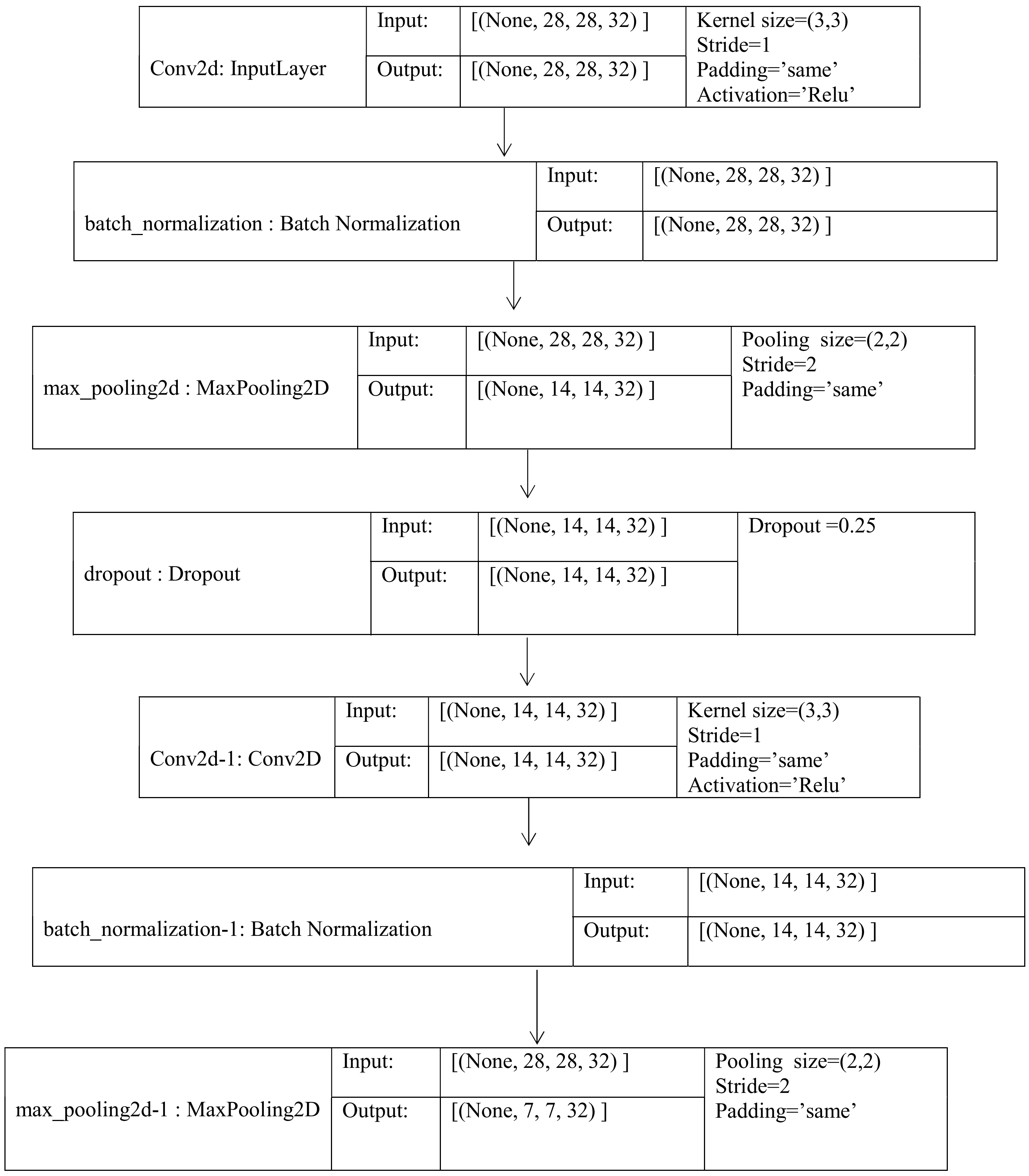


Figure 1. The proposed CNN's overall structure

- **Convolution Layers**

Five convolutional layer groups, five max-pooling layer groups, one fully connected layer, and a final softmax layer make up the new MCNN model. The activation function Relu, similar padding, stride one, a fixed kernel size of 3 by 3 (fixed), and the quantity of input and output channels in each convolutional layer make up the convolutional layer groups of our model. In addition, after every convolutional layer, a max-pooling layer is added, batch normalisation and RELU activation functions are applied, and Dropout is performed after every convolutional layer group. Then, a fully connected layer is included before softmax layers that flatten 2D spatial maps for image classification. Details of this model is shown in Figure 4.


Conv2d: InputLayer
Input: [(None, 28, 28, 32)]
Output: [(None, 28, 28, 32)]
Kernel size=(3,3)
Stride=1
Padding='same'
Activation='Relu'
batch_normalization : Batch Normalization
Input: [(None, 28, 28, 32)]
Output: [(None, 28, 28, 32)]
max_pooling2d : MaxPooling2D
Input: [(None, 28, 28, 32)]
Output: [(None, 14, 14, 32)]
Pooling size=(2,2)
Stride=2
Padding='same'
dropout : Dropout
Input: [(None, 14, 14, 32)]
Output: [(None, 14, 14, 32)]
Dropout =0.25
Conv2d-1: Conv2D
Input: [(None, 14, 14, 32)]
Output: [(None, 14, 14, 32)]
Kernel size=(3,3)
Stride=1
Padding='same'
Activation='Relu'
batch_normalization-1: Batch Normalization
Input: [(None, 14, 14, 32)]
Output: [(None, 14, 14, 32)]
max_pooling2d-1 : MaxPooling2D
Input: [(None, 28, 28, 32)]
Output: [(None, 7, 7, 32)]
Pooling size=(2,2)
Stride=2
Padding='same'

Dropout_1 : Dropout	Input:	[(None, 7, 7, 32)]	Dropout =0.25
	Output:	[(None, 7, 7, 32)]	

Conv2d-2: Conv2D	Input:	[(None, 7, 7, 64)]	Kernel size=(3,3) Stride=1 Padding='same' Activation='Relu'
	Output:	[(None, 7, 7, 64)]	

batch_normalization-2: Batch Normalization	Input:	[(None, 7, 7, 64)]
	Output:	[(None, 7, 7, 64)]

max_pooling2d-2 : MaxPooling2D	Input:	[(None, 7, 7, 64)]	Pooling size=(2,2) Stride=2 Padding='same'
	Output:	[(None, 4, 4, 64)]	

Dropout_2 : Dropout	Input:	[(None, 4, 4, 64]	Dropout =0.25
	Output:	[(None, 4, 4, 64)]	

Conv2d-3: Conv2D	Input:	[(None, 4, 4, 64)]	Kernel size=(3,3) Stride=1 Padding='same' Activation='Relu'
	Output:	[(None, 4, 4, 64)]	

batch_normalization-3: Batch Normalization	Input:	[(None, 4, 4, 64)]
	Output:	[(None, 4, 4, 64)]

max_pooling2d-3 : MaxPooling2D	Input:	[(None, 4, 4, 64)]	Kernel size=(2,2)
			Stride=2
			Padding='same'

	Output:	[(None, 2, 2, 64)]	

↓

Dropout_3 : Dropout	Input:	[(None, 2, 2, 64]	Dropout =0.25
	Output:	[(None, 2, 2, 64)]	

↓

Conv2d-4: Conv2D	Input:	[(None, 2, 2, 64)]	Kernel size=(3,3)
			Stride=1
			Padding='same'
			Activation='Relu'

	Output:	[(None, 2, 2, 128)]	

↓

batch_normalization-4: Batch Normalization	Input:	[(None, 2, 2, 128)]
	Output:	[(None, 2, 2, 128)]

↓

max_pooling2d-4 : MaxPooling2D	Input:	[(None, 2, 2, 128)]	Kernel size=(2,2)
			Stride=2
			Padding='same'

	Output:	[(None, 1, 1, 128)]	

↓

Dropout_4 : Dropout	Input:	[(None, 1, 1, 128]	Dropout =0.25
	Output:	[(None, 1, 1, 128)]	

↓

flatten : Flatten	Input:	[(None, 1, 1, 128]
	Output:	[(None, 128)]

↓

	Input:	[(None, 128)]	Activation='Relu'

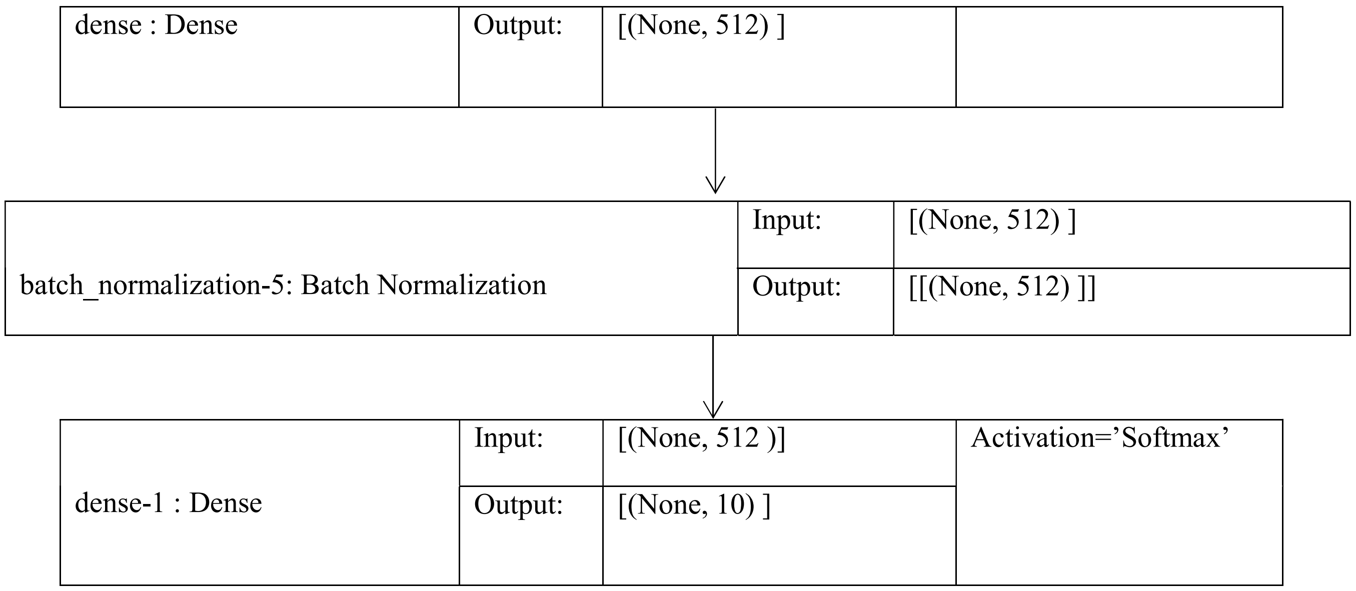
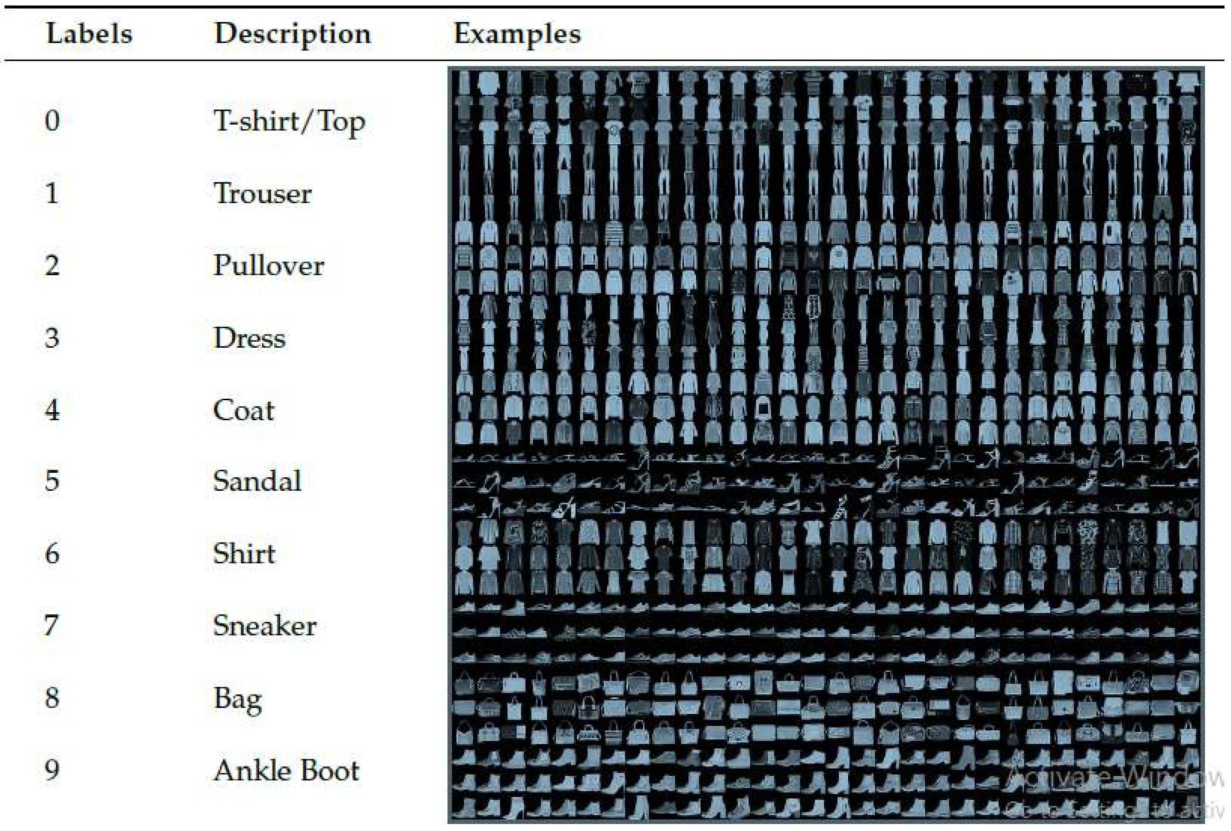


Figure 2. CNN Work flow

2- Experiment and Result

- Dataset



Labels	Description	Examples
0	T-shirt/Top	
1	Trouser	
2	Pullover	
3	Dress	
4	Coat	
5	Sandal	
6	Shirt	
7	Sneaker	
8	Bag	
9	Ankle Boot	

Figure 3. Fashion-MNIST dataset

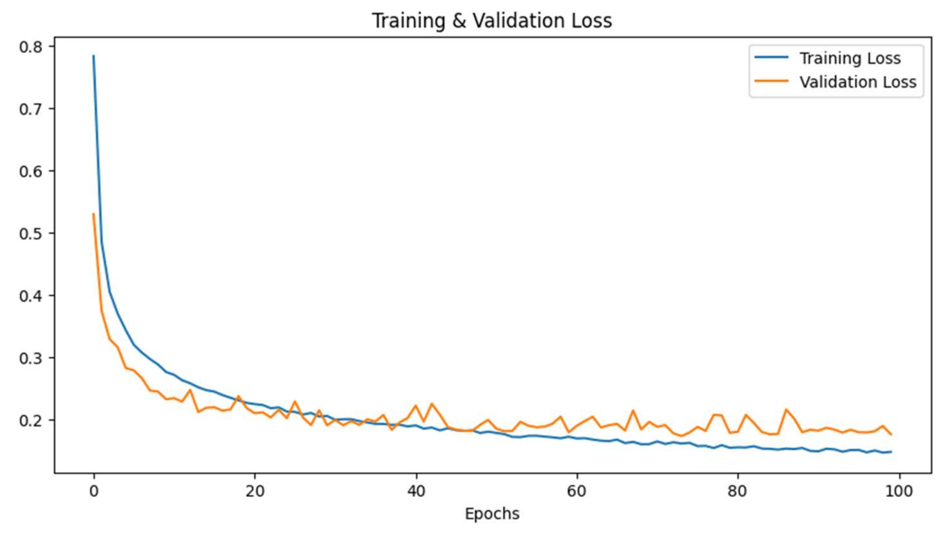


Figure 4. Loss of training and validation

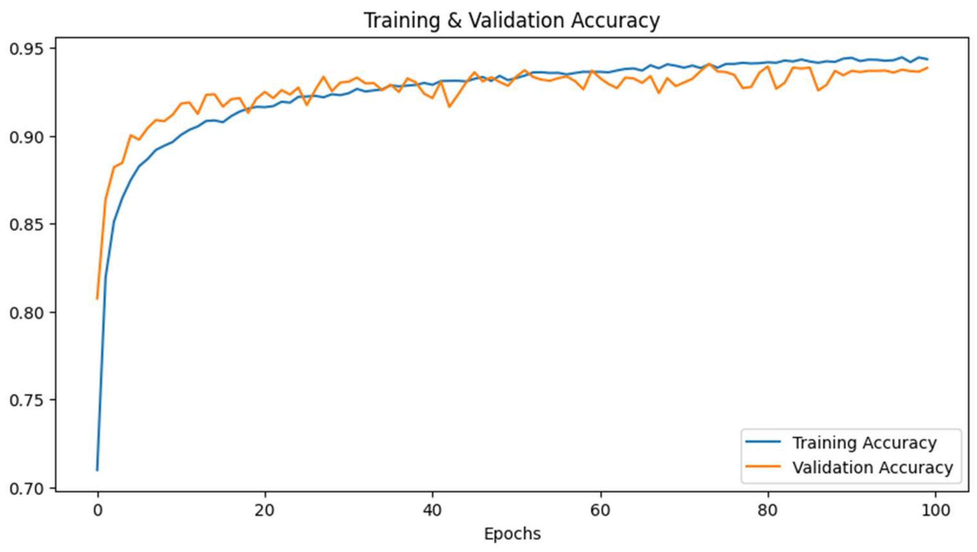


Figure 5. Accuracy of training and validation

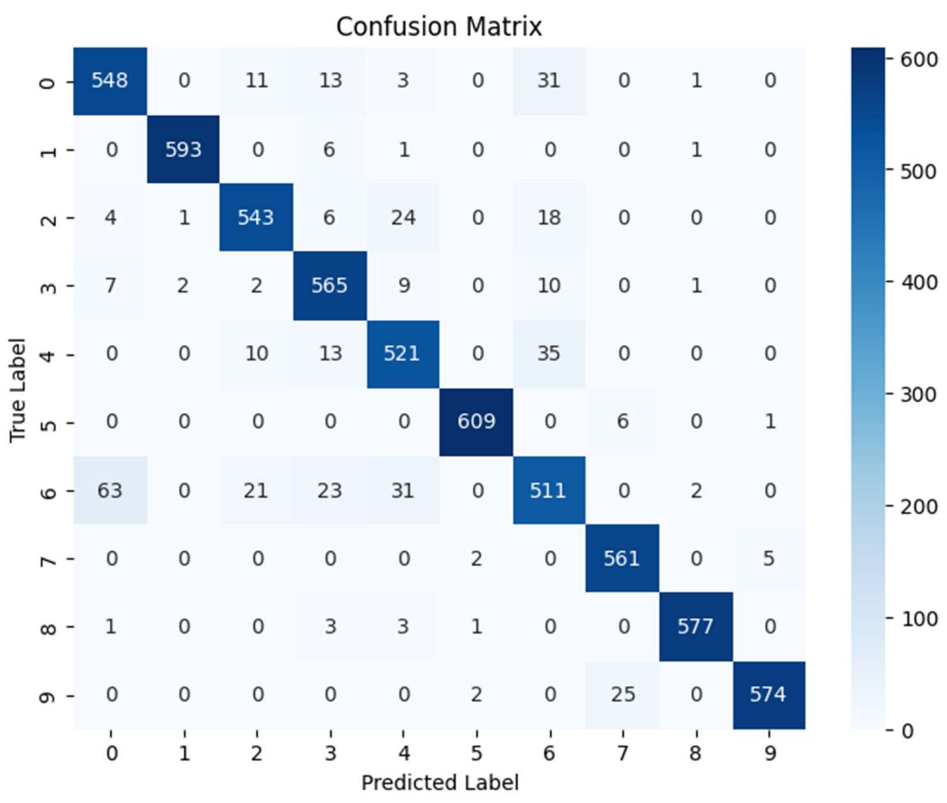


Figure 6. Confusion matrix

Table 1. CNN Classification Report

	Precision	Recall	F1-score	Support
T-shirt/top	0.8796	0.9028	0.8911	607
Trouser	0.9950	0.9867	0.9908	601
Pullover	0.9250	0.9111	0.9180	596
Dress	0.8983	0.9480	0.9224	596
Coat	0.8801	0.8998	0.8898	579
Sandal	0.9919	0.9886	0.9902	616
Shirt	0.8446	0.7849	0.8137	651

Sneaker	0.9476	0.9877	0.9672	568
Bag	0.9914	0.9863	0.9889	585
Ankle boot	0.9897	0.9551	0.9721	601

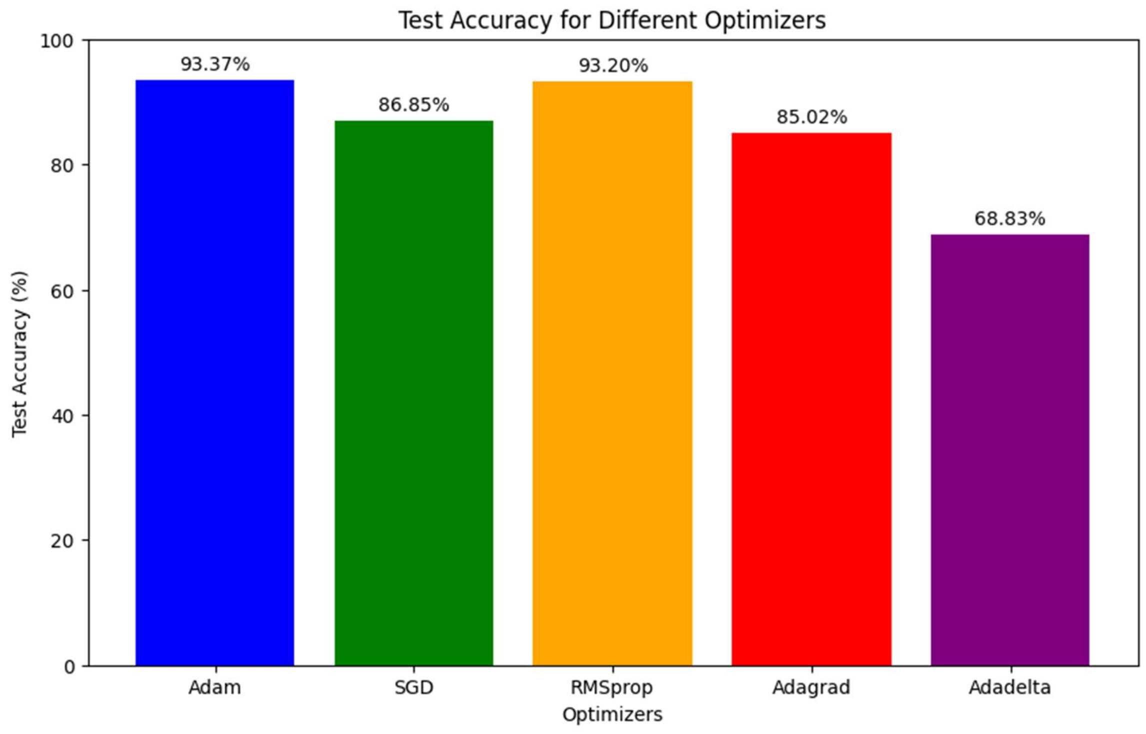


Figure 7. Performance of different optimizers

Table 2. Compression of different batch size

Batch size	**Train accuracy (%)**	**Validation accuracy (%)**	**Test accuracy (%)**
16	94.09	93.38	93.02
32	94.18	93.67	93.28
64	94.29	93.77	93.12
128	94.65	92.32	93.37
256	94.58	93.67	93.35

Table 3. Compression of different learning rate

Learning rate	**Train accuracy (%)**	**Validation accuracy (%)**	**Test accuracy (%)**
0.01	94.04	93.55	93.27
0.02	93.42	93.17	93.05
0.001	94.65	92.32	93.37
0.002	94.45	93.62	93.18

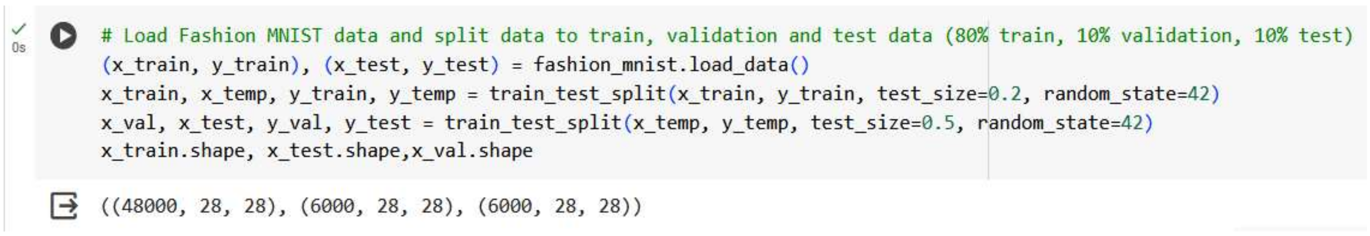
Table 4. Compare different model

	epoch	**Train accuracy %**	**Validation accuracy %**	**Test accuracy %**
ResNet50	100	83.30	82.96	82.70
VGG16	100	89.66	91.10	90.54
Proposed method	100	**94.65**	**92.32**	**93.37**

3- Coding details

I have run my code on Google Colab using a GPU. Below code loads a Fashion Mnist dataset and splits it into train, validation, and test data.

- Load dataset



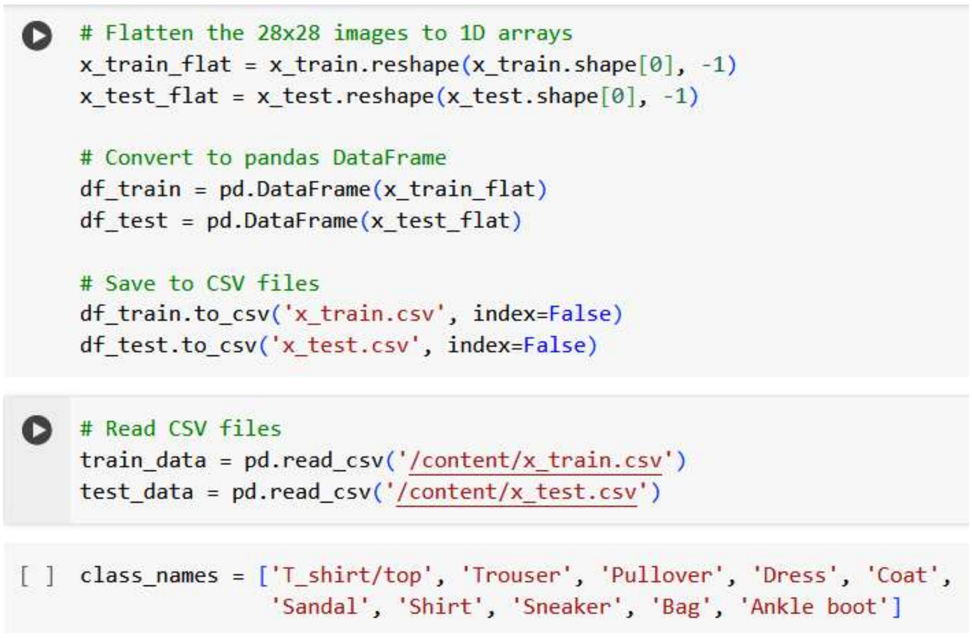
```
# Load Fashion MNIST data and split data to train, validation and test data (80% train, 10% validation, 10% test)
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
x_train, x_temp, y_train, y_temp = train_test_split(x_train, y_train, test_size=0.2, random_state=42)
x_val, x_test, y_val, y_test = train_test_split(x_temp, y_temp, test_size=0.5, random_state=42)
x_train.shape, x_test.shape,x_val.shape
```

```
((48000, 28, 28), (6000, 28, 28), (6000, 28, 28))
```

Figure 8. Load and split dataset

- Visualization

Figure 12 shows the source code of the visualization, and Figure 13 shows the first 25 images from the training set and displays the class name below each image. First, convert the 28×28 image to 1D arrays and convert them to a Pandas dataframe for saving to CSV files, and then these files are read. Finally, we shows the first 25 images from the training set using the below code.



```
# Flatten the 28x28 images to 1D arrays
x_train_flat = x_train.reshape(x_train.shape[0], -1)
x_test_flat = x_test.reshape(x_test.shape[0], -1)

# Convert to pandas DataFrame
df_train = pd.DataFrame(x_train_flat)
df_test = pd.DataFrame(x_test_flat)

# Save to CSV files
df_train.to_csv('x_train.csv', index=False)
df_test.to_csv('x_test.csv', index=False)
```

```
# Read CSV files
train_data = pd.read_csv('/content/x_train.csv')
test_data = pd.read_csv('/content/x_test.csv')
```

```
[ ] class_names = ['T_shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
                   'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

Figure 9. Preprocessing for visualization

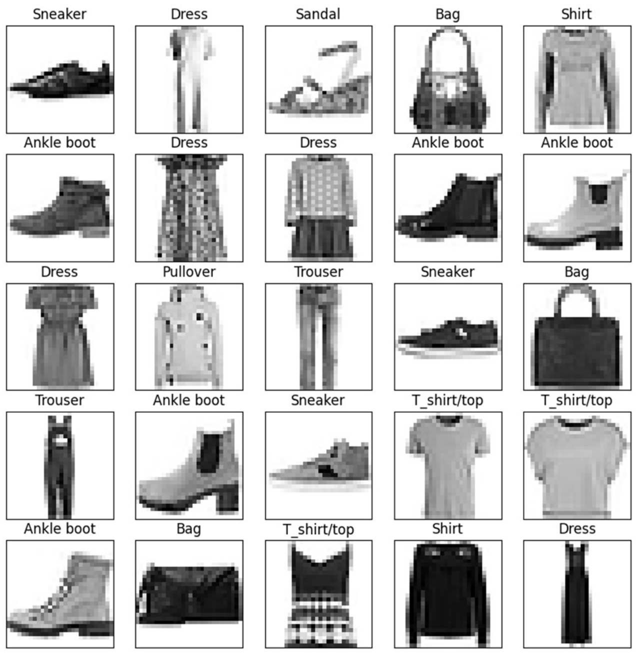
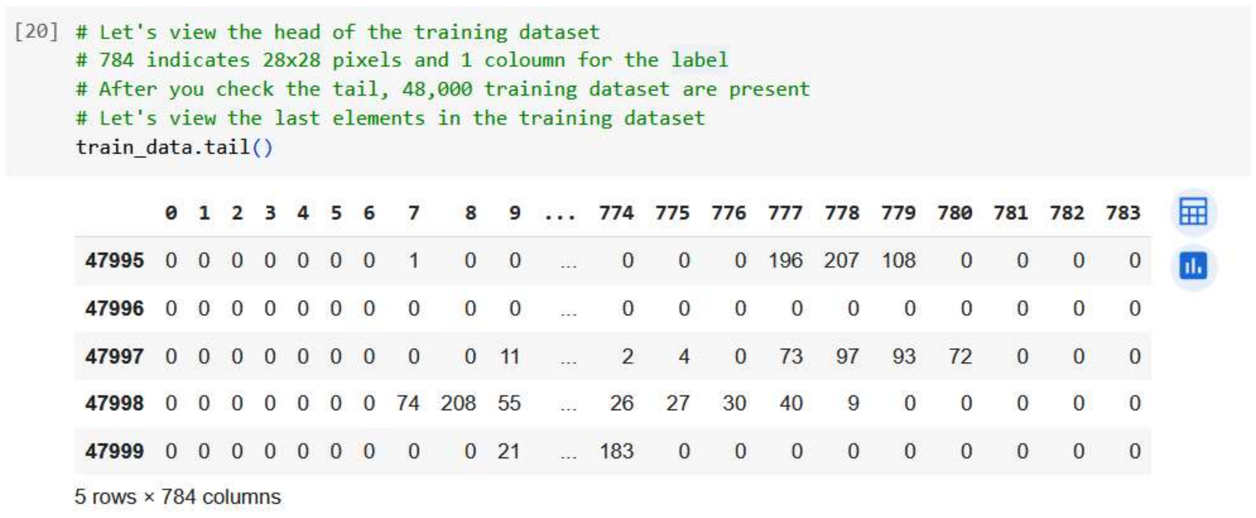


Figure10. Display the first 25 images from the training set

Figure 11 shows the latest elements in the training dataset. 48,000 training dataset are present and 784 indicates 28×28 pixels and 1 column for the label.



```
[20] # Let's view the head of the training dataset
     # 784 indicates 28x28 pixels and 1 coloumn for the label
     # After you check the tail, 48,000 training dataset are present
     # Let's view the last elements in the training dataset
     train_data.tail()
```

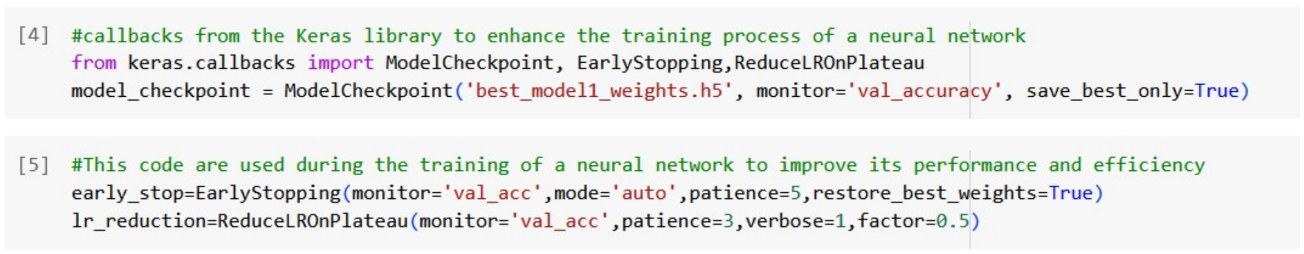
	0	1	2	3	4	5	6	7	8	9	...	774	775	776	777	778	779	780	781	782	783
47995	0	0	0	0	0	0	0	1	0	0	...	0	0	0	196	207	108	0	0	0	0
47996	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
47997	0	0	0	0	0	0	0	0	0	11	...	2	4	0	73	97	93	72	0	0	0
47998	0	0	0	0	0	0	0	74	208	55	...	26	27	30	40	9	0	0	0	0	0
47999	0	0	0	0	0	0	0	0	0	21	...	183	0	0	0	0	0	0	0	0	0

5 rows × 784 columns

Figure 11. The latest elements in the training dataset

- Explanation of Proposed model

The below code is using callbacks from the Keras library to enhance the training process of a neural network. 'ModelCheckpoint' saves the model weights to a file whenever the validation accuracy improves. 'EarlyStopping' stops training if the monitored quantity has not increased. When a metric stops improving, "ReduceLROnPlateau" lowers the learning rate at that point. (Figure 12).

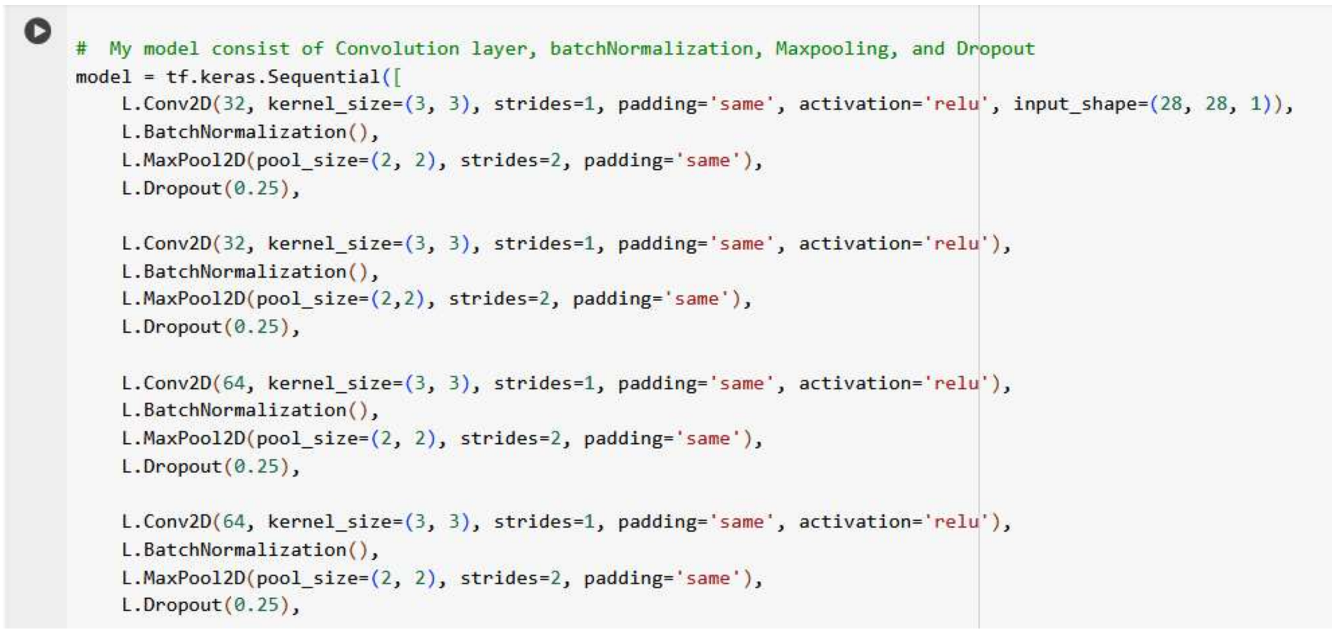


```
[4] #callbacks from the Keras library to enhance the training process of a neural network
    from keras.callbacks import ModelCheckpoint, EarlyStopping,ReduceLROnPlateau
    model_checkpoint = ModelCheckpoint('best_model1_weights.h5', monitor='val_accuracy', save_best_only=True)
```

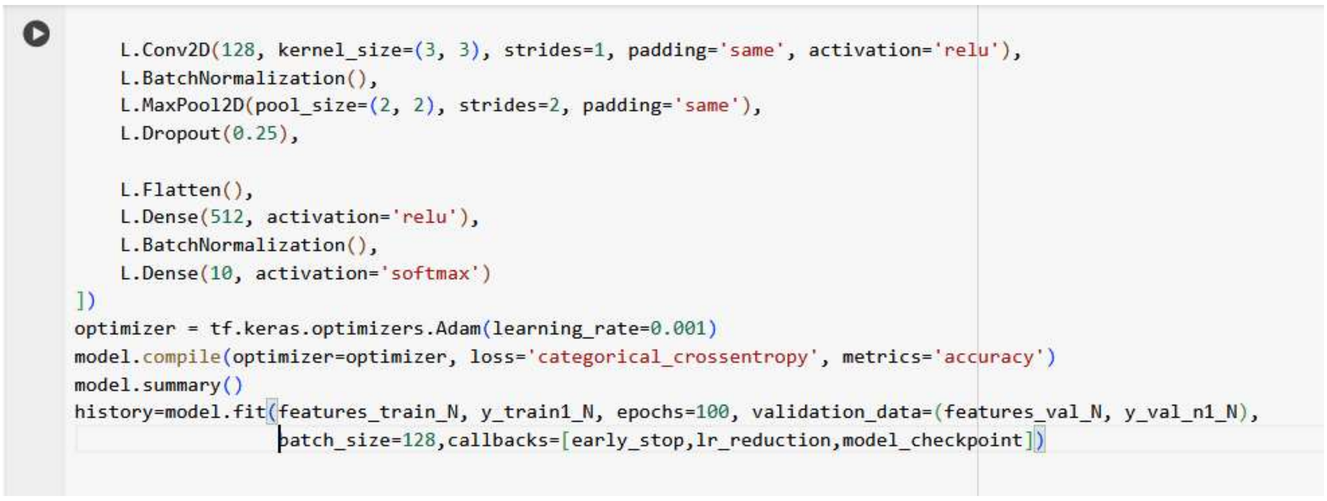
```
[5] #This code are used during the training of a neural network to improve its performance and efficiency
    early_stop=EarlyStopping(monitor='val_acc',mode='auto',patience=5,restore_best_weights=True)
    lr_reduction=ReduceLROnPlateau(monitor='val_acc',patience=3,verbose=1,factor=0.5)
```

Figure 12. Enhance the training process of a neural network

Figure 13 shows proposed model and its details and also figure 14 displays summary of proposed model.

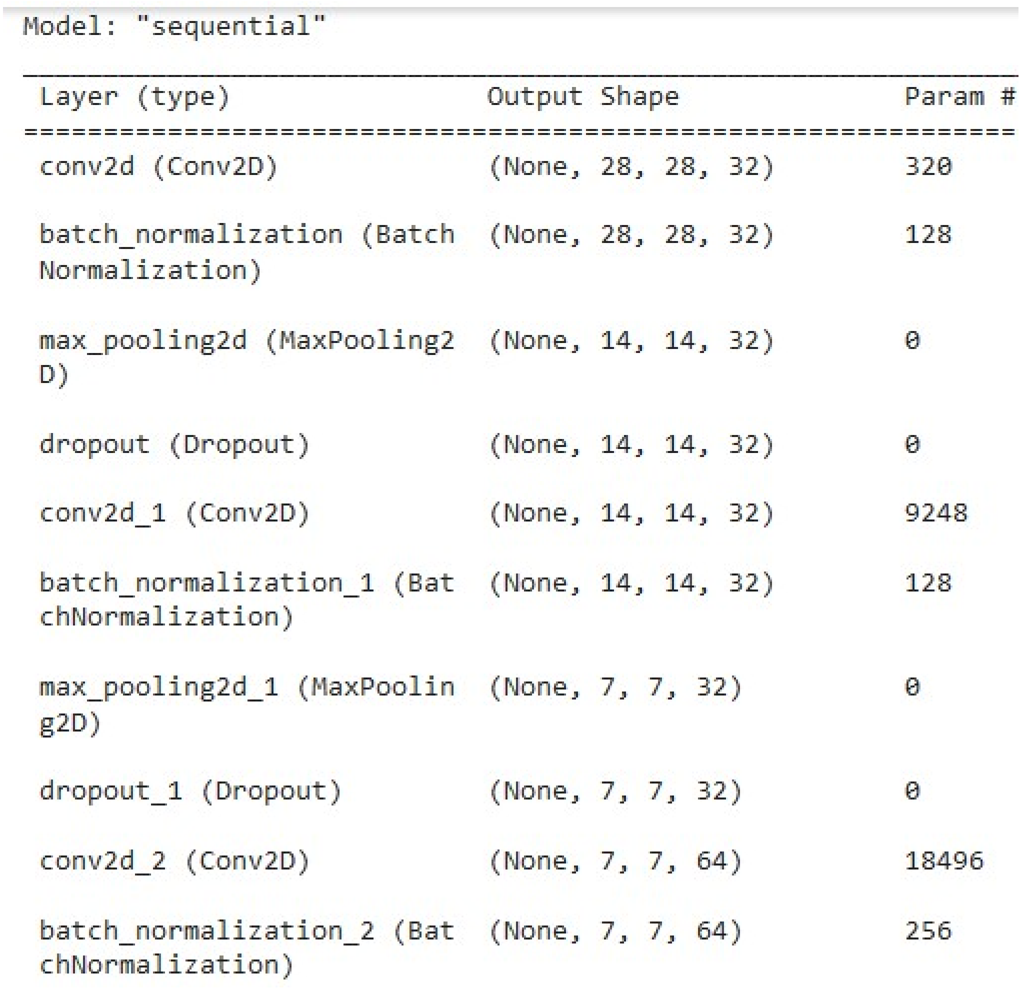


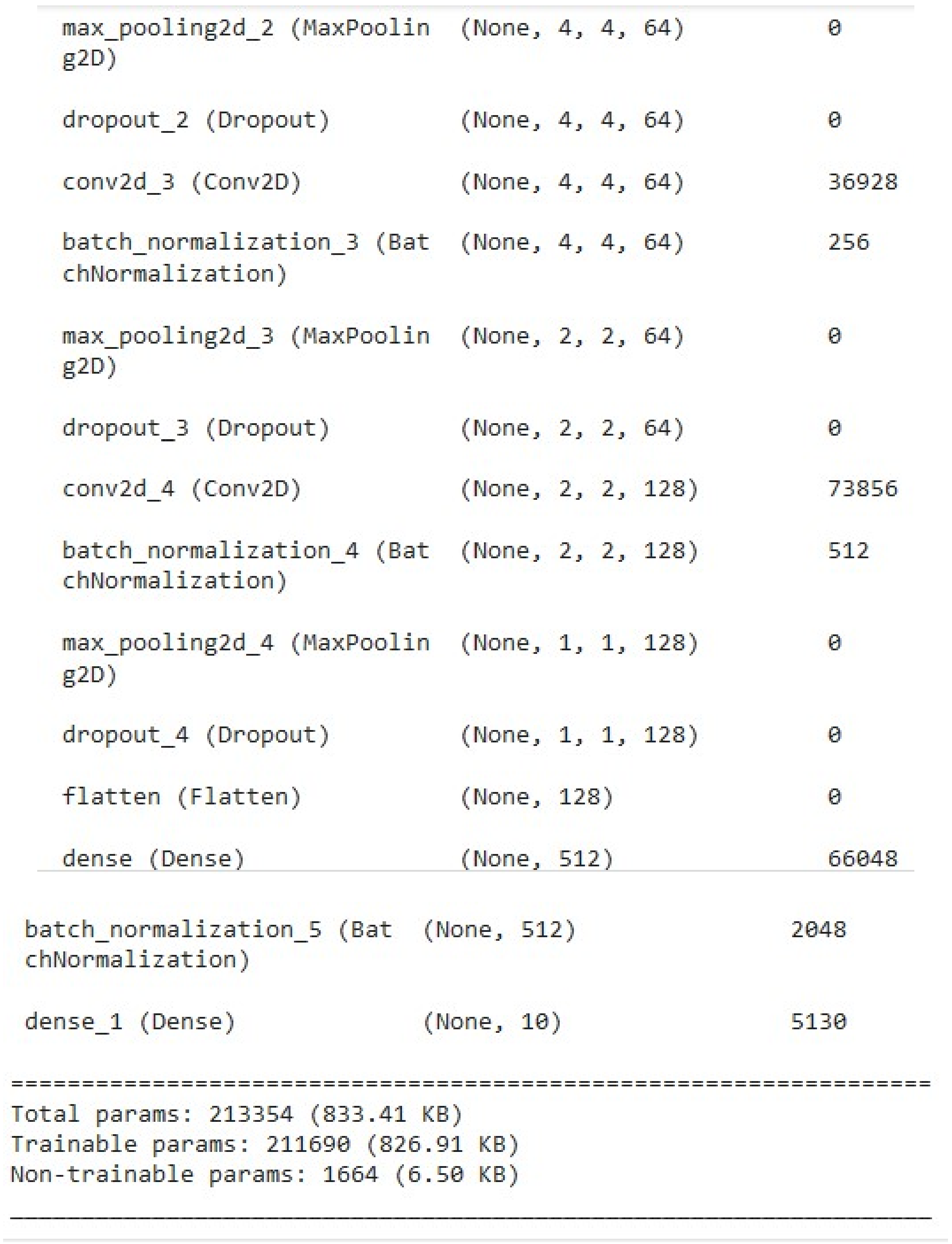
```
#  My model consist of Convolution layer, batchNormalization, Maxpooling, and Dropout
model = tf.keras.Sequential([
    L.Conv2D(32, kernel_size=(3, 3), strides=1, padding='same', activation='relu', input_shape=(28, 28, 1)),
    L.BatchNormalization(),
    L.MaxPool2D(pool_size=(2, 2), strides=2, padding='same'),
    L.Dropout(0.25),

    L.Conv2D(32, kernel_size=(3, 3), strides=1, padding='same', activation='relu'),
    L.BatchNormalization(),
    L.MaxPool2D(pool_size=(2,2), strides=2, padding='same'),
    L.Dropout(0.25),

    L.Conv2D(64, kernel_size=(3, 3), strides=1, padding='same', activation='relu'),
    L.BatchNormalization(),
    L.MaxPool2D(pool_size=(2, 2), strides=2, padding='same'),
    L.Dropout(0.25),

    L.Conv2D(64, kernel_size=(3, 3), strides=1, padding='same', activation='relu'),
    L.BatchNormalization(),
    L.MaxPool2D(pool_size=(2, 2), strides=2, padding='same'),
    L.Dropout(0.25),
```



```
    L.Conv2D(128, kernel_size=(3, 3), strides=1, padding='same', activation='relu'),
    L.BatchNormalization(),
    L.MaxPool2D(pool_size=(2, 2), strides=2, padding='same'),
    L.Dropout(0.25),

    L.Flatten(),
    L.Dense(512, activation='relu'),
    L.BatchNormalization(),
    L.Dense(10, activation='softmax')
])
optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics='accuracy')
model.summary()
history=model.fit(features_train_N, y_train1_N, epochs=100, validation_data=(features_val_N, y_val_n1_N),
                  batch_size=128,callbacks=[early_stop,lr_reduction,model_checkpoint])
```

Figure 13. Proposed model


```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 32)        320

 batch_normalization (Batch  (None, 28, 28, 32)        128
 Normalization)

 max_pooling2d (MaxPooling2  (None, 14, 14, 32)        0
 D)

 dropout (Dropout)           (None, 14, 14, 32)        0

 conv2d_1 (Conv2D)           (None, 14, 14, 32)        9248

 batch_normalization_1 (Bat  (None, 14, 14, 32)        128
 chNormalization)

 max_pooling2d_1 (MaxPoolin  (None, 7, 7, 32)          0
 g2D)

 dropout_1 (Dropout)         (None, 7, 7, 32)          0

 conv2d_2 (Conv2D)           (None, 7, 7, 64)          18496

 batch_normalization_2 (Bat  (None, 7, 7, 64)          256
 chNormalization)
```

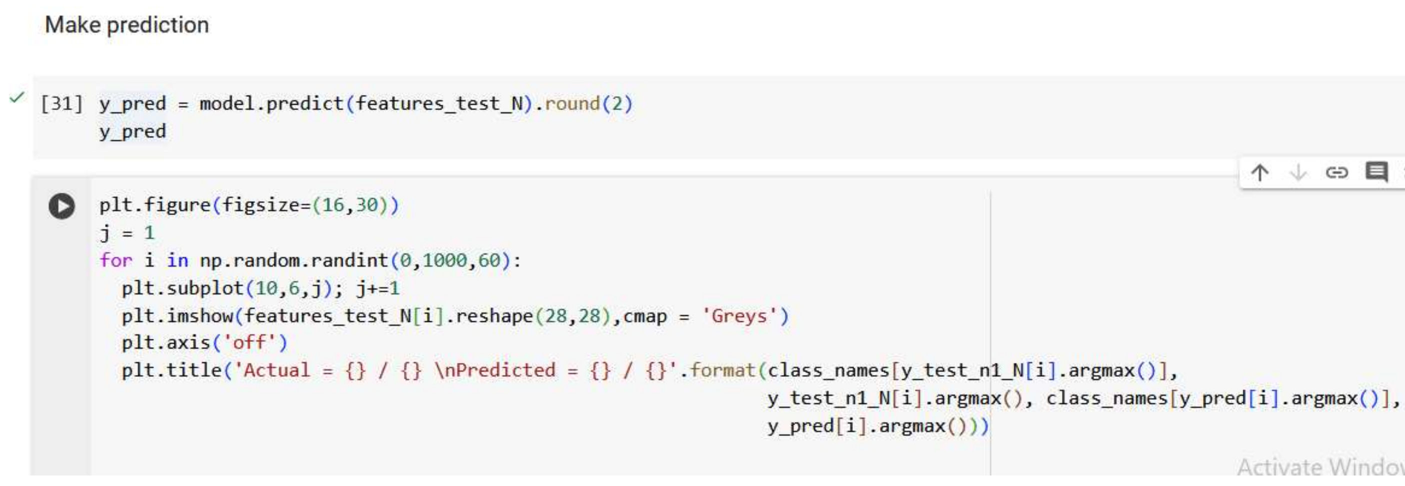


```
 max_pooling2d_2 (MaxPoolin  (None, 4, 4, 64)          0
 g2D)

 dropout_2 (Dropout)         (None, 4, 4, 64)          0

 conv2d_3 (Conv2D)           (None, 4, 4, 64)          36928

 batch_normalization_3 (Bat  (None, 4, 4, 64)          256
 chNormalization)

 max_pooling2d_3 (MaxPoolin  (None, 2, 2, 64)          0
 g2D)

 dropout_3 (Dropout)         (None, 2, 2, 64)          0

 conv2d_4 (Conv2D)           (None, 2, 2, 128)         73856

 batch_normalization_4 (Bat  (None, 2, 2, 128)         512
 chNormalization)

 max_pooling2d_4 (MaxPoolin  (None, 1, 1, 128)         0
 g2D)

 dropout_4 (Dropout)         (None, 1, 1, 128)         0

 flatten (Flatten)           (None, 128)               0

 dense (Dense)               (None, 512)               66048

 batch_normalization_5 (Bat  (None, 512)               2048
 chNormalization)

 dense_1 (Dense)             (None, 10)                5130

=================================================================
Total params: 213354 (833.41 KB)
Trainable params: 211690 (826.91 KB)
Non-trainable params: 1664 (6.50 KB)
_________________________________________________________________
```

Figure 14. Summary of proposed model

- Make prediction

Figure 15 is a snippet for creating a grid of subplots to visualize a random sample of images from a test set along with their corresponding actual and predicted labels. Result is shown in figure 16. This code creates a grid of subplots that shows the actual and expected labels for each image in a random sample of sixty images taken from a test set. This can be helpful for visually examining how well a machine learning model performs on certain test data.


Make prediction

```
[31] y_pred = model.predict(features_test_N).round(2)
     y_pred
```

```
plt.figure(figsize=(16,30))
j = 1
for i in np.random.randint(0,1000,60):
  plt.subplot(10,6,j); j+=1
  plt.imshow(features_test_N[i].reshape(28,28),cmap = 'Greys')
  plt.axis('off')
  plt.title('Actual = {} / {} \nPredicted = {} / {}'.format(class_names[y_test_n1_N[i].argmax()],
                                                            y_test_n1_N[i].argmax(), class_names[y_pred[i].argmax()],
                                                            y_pred[i].argmax()))
```

Figure 15. Prediction source code

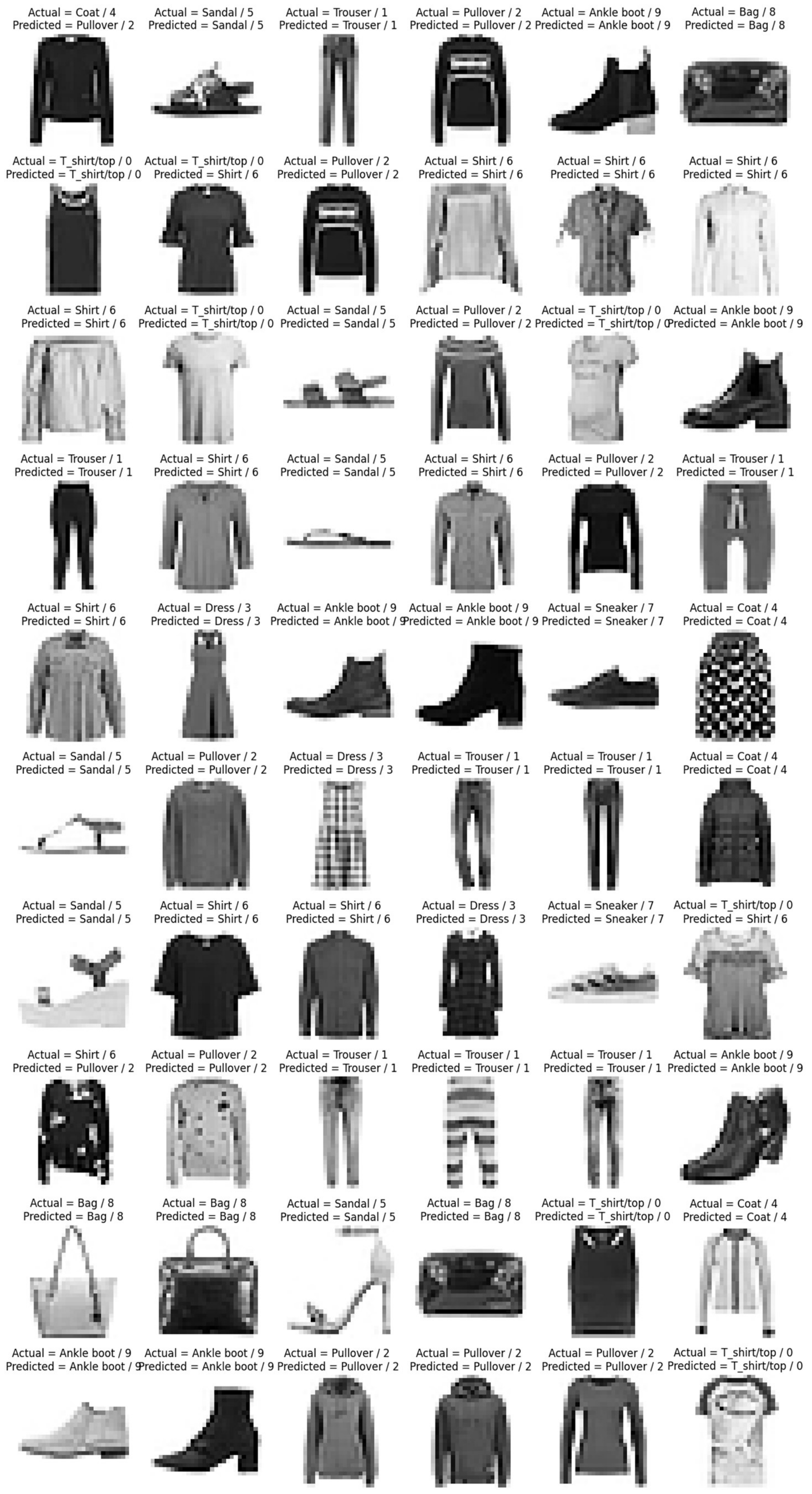


Figure 16. Random sample of images from a test set along with their corresponding actual and predicted labels

3- Future work

In the future, we want to enhance the accuracy of our model in the future by adjusting the layer typology and some internal parameters. To increase the accuracy of the architecture, we would also like to create a new model for classifying fashion images by merging various already-existing methodologies. We want to examine this in comparison to other datasets. One dataset with low-resolution photos is Fashion MNIST. In the future, we hope to test CNN architecture using a dataset of actual clothing images that we have collected, in addition to trying these high-resolution images.