FASHION MNIST CLASSIFICATION

BY: TWISHA KAUL

ABSTRACT

In this project, we have built a fashion apparel recognition using the Convolutional Neural Network (CNN) model. To train the CNN model, we have used the Fashion MNIST dataset. After successful training, the CNN model can predict the name of the class given apparel item belongs to. This is a multiclass classification problem in which there are 10 apparel classes the items will be classified.

The fashion training set consists of 70,000 images divided into 60,000 training and 10,000 testing samples. The dataset sample consists of 28x28 grayscale images, associated with a label from 10 classes.

So the end goal is to train and test the model using a Convolution neural network(CNN)

OBJECTIVE

The objective is to identify(predict) different fashion products from given images using CNN.

The 'target' dataset has 10 class labels, (0 – T-shirt/top, 1 – Trouser,....9 – Ankle Boot).

From the given images, we need to classify them into one of these classes, hence, it is essentially a 'Multi-class Classification' problem.

INTRODUCTION

What is CNN?

• Convolutional neural networks, like neural networks, consisting of neurons with learnable weights and biases. Each neuron receives multiple inputs, takes a weighted sum over them, passes them through an activation function, and responds with an output.

• The entire network has an activation function loss and all the tips and tricks that we developed for Neural Networks also apply to Convolutional Neural Networks.

• Neural Networks, as the name suggests, is a machine learning technique that is modeled after the Brain. Structure. It consists of a network of learning units called neurons.

• These neurons learn to convert input signals (e.g. an image of a cat) into corresponding output signals (e.g. an image of a cat). cat"), which form the basis for automatic detection.

Example:

• Let’s take the example of automatic image recognition. The process of determining whether an image contains a cat, which involves an activation function. If the image resembles images of previous cats that the neurons have seen before, the tag "cat" would be shown.

• So, the more labeled images the neurons are exposed to, the better it learns to recognize other unlabeled images. We call this the process of neuron formation.

How does CNN work?

There are four layered concepts in Convolutional Neural Networks(CNN):

1. Convolution,
2. Relu,
3. Pooling, and
4. Full Connectedness (Fully Connected Layer).

1. Convolution

Each convolution filter represents a feature of interest (e.g. pixels in letters) and the convolutional neural network algorithm learns which features comprise the resulting reference (ie the alphabet).

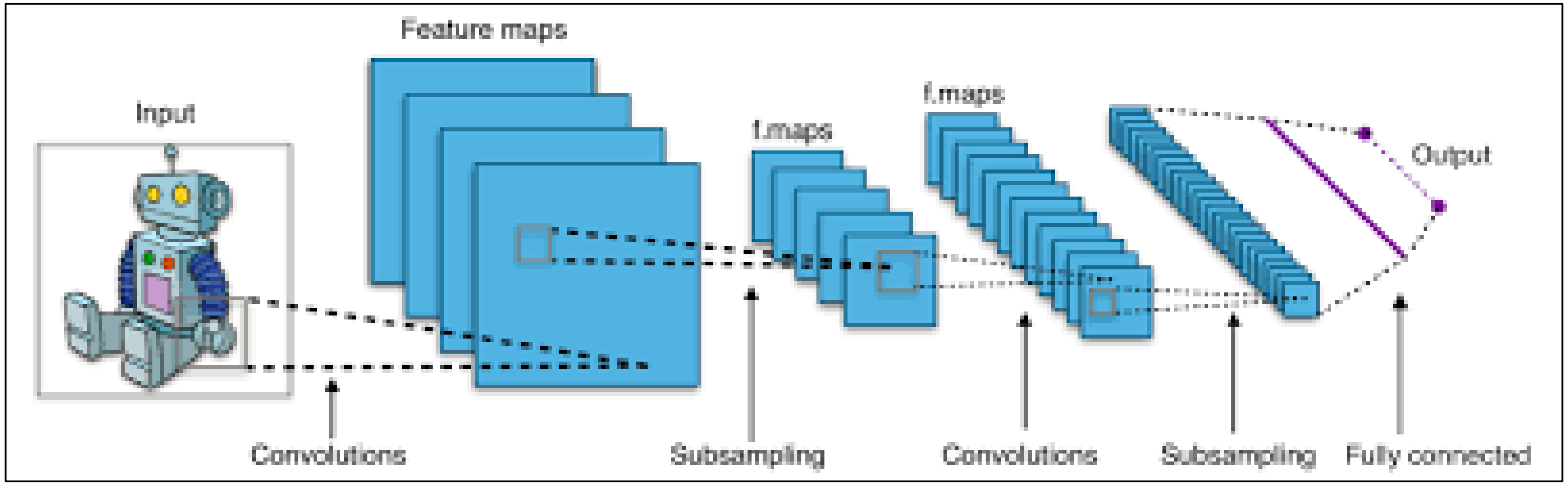
2. Relu

Relu removes all negative values from the convolution. All positive values remain the same, but all negative values are changed to zero.

3. Pooling

In this layer, we shrink the image stack to a smaller size. The grouping takes place after passing through the activation layer. For this we implement the following 4 steps:

- Choose a window size (usually 2 or 3)
- Choose one step (usually 2)
- Step through your window through your filtered images
- Take in each window the maximum value



METHODOLOGY

<u>Phases:</u>

1. Data pre-processing
2. Build CNN
3. Test and Evaluate
4. Saving

DATA PRE-PROCESSING

- Importing libraries
- Loading data
- Showing images from numbers
- Changing dimensions (3D-4D)

```
#import libraries

import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import keras
import tensorflow as tf

#load data

(X_train, y_train), (X_test, y_test) = tf.keras.datasets.fashion_mnist.load_data()

X_train.shape, y_train.shape      #X_train = 60000 images and each image is 28/28 pixel
                                  #y_train = 60000 levels

    ((60000, 28, 28), (60000,))

X_test.shape, y_test.shape

    ((10000, 28, 28), (10000,))

#seeing datasets

X_train

    array([[[0, 0, 0, ..., 0, 0, 0],
            [0, 0, 0, ..., 0, 0, 0],
            [0, 0, 0, ..., 0, 0, 0],
            ...,
            [0, 0, 0, ..., 0, 0, 0],
            [0, 0, 0, ..., 0, 0, 0],
            [0, 0, 0, ..., 0, 0, 0]],

           [[0, 0, 0, ..., 0, 0, 0],
            [0, 0, 0, ..., 0, 0, 0],
            [0, 0, 0, ..., 0, 0, 0],
            ...,
            [0, 0, 0, ..., 0, 0, 0],
```

BUILDING FIRST CNN

- Input/Image
- First Convolutional Layer
- Max pulling layer
- Flatten
- First input layer
- Last output layer

6. Last output layer

```
model = keras.models.Sequential([
    keras.layers.Conv2D(filters = 32, kernel_size = 3, strides = (1,1), padding = 'valid',  activation = 'relu', input_shape = [28,28,
    # strides - how many columns to skip, padding - to add any extra column or row, relu - will give 0 for negative or 0 values.
    keras.layers.MaxPooling2D(pool_size=(2,2)),
    keras.layers.Flatten(),    #flatten all data into single vector
    keras.layers.Dense(units = 128, activation = 'relu'),
    keras.layers.Dense(units = 10, activation = 'softmax')
])
```

```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 26, 26, 32)        320

 max_pooling2d_2 (MaxPooling  (None, 13, 13, 32)       0
 2D)

 flatten_2 (Flatten)         (None, 5408)              0

 dense_4 (Dense)             (None, 128)               692352

 dense_5 (Dense)             (None, 10)                1290

=================================================================
Total params: 693,962
Trainable params: 693,962
Non-trainable params: 0
_________________________________________________________________
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 26, 26, 32)        320

 max_pooling2d_3 (MaxPooling  (None, 13, 13, 32)       0
 2D)

 flatten_3 (Flatten)         (None, 5408)              0

 dense_6 (Dense)             (None, 128)               692352

 dense_7 (Dense)             (None, 10)                1290
```

TEST AND EVALUATE

```
# Testing Model
```

```
model.predict(np.expand_dims(X_test[0], axis = 0)).round(2)
```

```
1/1 [==============================] - 0s 24ms/step
array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 1.]], dtype=float32)
```

```
np.argmax(model.predict(np.expand_dims(X_test[0],axis = 0)).round(2))
```

```
1/1 [==============================] - 0s 22ms/step
9
```

```
y_test[0]
```

```
9
```

```
y_test
```

```
array([9, 2, 1, ..., 8, 1, 5], dtype=uint8)
```

```
y_pred = model.predict(X_test)
y_pred.round(2)
```

```
313/313 [==============================] - 2s 8ms/step
array([[0.  , 0.  , 0.  , ..., 0.  , 0.  , 1.  ],
       [0.  , 0.  , 1.  , ..., 0.  , 0.  , 0.  ],
       [0.  , 1.  , 0.  , ..., 0.  , 0.  , 0.  ],
       ...,
       [0.  , 0.  , 0.  , ..., 0.  , 1.  , 0.  ],
       [0.  , 1.  , 0.  , ..., 0.  , 0.  , 0.  ],
       [0.  , 0.  , 0.  , ..., 0.05, 0.  , 0.  ]], dtype=float32)
```

```
model.evaluate(X_test, y_test)
```

```
313/313 [==============================] - 3s 8ms/step - loss: 0.3839 - accuracy: 0.8852
[0.3839433193206787, 0.885200023651123]
```

SAVING

```
# Saving model
```

model name.save(path)

If u need to save it in google colab only then, just write the name of the model in the ( )

If you want to save your project in google drive then, follow the same step as above and then in the folder region click on the drive icon and mout your file in google drive.

```
model.save('Fashion_MNIST_Classification.h5')  # h5 is an extension here
```

```
# for saving in google drive
path = '/content/drive/MyDrive/Fashion_MNIST_Classification.h5'
model.save(path)
```

```
# Deployment (in google colab)
```

```
#loading model
model_deploy = keras.models.load_model('Fashion_MNIST_Classification.h5')
```

BUILD 2 COMPLEX CNN

1 st complex model

- Input/Image
- First convolutional layer
- First max pooling layer
- Second convolutional layer
- Second max pooling layer
- Flatten
- First input layer
- First hidden layer
- Second hidden layer
- Last output layer

```
model_2 = keras.models.Sequential([
                         keras.layers.Conv2D(filters=32, kernel_size=3, strides=(1,1), padding='valid',activation= 'relu', input_shap
                         keras.layers.MaxPooling2D(pool_size=(2,2)),
                         keras.layers.Conv2D(filters=64, kernel_size=3, strides=(2,2), padding='same', activation='relu'),
                         keras.layers.MaxPooling2D(pool_size=(2,2)),
                         keras.layers.Flatten(),
                         keras.layers.Dense(units=128, activation='relu'),   #first input layer
                         keras.layers.Dropout(0.25),        # this will regularize our model
                         keras.layers.Dense(units=256, activation='relu'),   # first hidden layer
                         keras.layers.Dropout(0.25),
                         keras.layers.Dense(units=128, activation='relu'),    # second hidden layer
                         keras.layers.Dense(units=10, activation='softmax')    #last output layer
                         ])
```

```
#compiling model
model_2.compile(optimizer = 'adam', loss = 'sparse_categorical_crossentropy', metrics = ['accuracy'])
```

```
# Training model
model_2.fit(X_train, y_train, epochs = 20, batch_size = 512, verbose = 1, validation_data = (X_val, y_val))
```

```
Epoch 1/20
94/94 [==============================] - 28s 280ms/step - loss: 2.0244 - accuracy: 0.5850 - val_loss: 0.5536 - val_accuracy: 0.7
Epoch 2/20
94/94 [==============================] - 27s 291ms/step - loss: 0.5929 - accuracy: 0.7799 - val_loss: 0.4583 - val_accuracy: 0.8
Epoch 3/20
94/94 [==============================] - 34s 361ms/step - loss: 0.4972 - accuracy: 0.8182 - val_loss: 0.4096 - val_accuracy: 0.8
Epoch 4/20
94/94 [==============================] - 26s 278ms/step - loss: 0.4455 - accuracy: 0.8365 - val_loss: 0.3908 - val_accuracy: 0.8
Epoch 5/20
94/94 [==============================] - 26s 276ms/step - loss: 0.4086 - accuracy: 0.8485 - val_loss: 0.3901 - val_accuracy: 0.8
Epoch 6/20
94/94 [==============================] - 26s 275ms/step - loss: 0.3891 - accuracy: 0.8569 - val_loss: 0.3651 - val_accuracy: 0.8
Epoch 7/20
94/94 [==============================] - 26s 277ms/step - loss: 0.3628 - accuracy: 0.8672 - val_loss: 0.3491 - val_accuracy: 0.8
Epoch 8/20
94/94 [==============================] - 27s 290ms/step - loss: 0.3466 - accuracy: 0.8720 - val_loss: 0.3381 - val_accuracy: 0.8
```

2nd complex model

```
94/94 [==============================] - 56s 598ms/step - loss: 0.3138 - accuracy: 0.8885 - val_loss: 0.3400 - val_accuracy:
Epoch 8/50
94/94 [==============================] - 56s 598ms/step - loss: 0.2968 - accuracy: 0.8951 - val_loss: 0.3173 - val_accuracy:
Epoch 9/50
94/94 [==============================] - 56s 598ms/step - loss: 0.2776 - accuracy: 0.9014 - val_loss: 0.3064 - val_accuracy:
Epoch 10/50
94/94 [==============================] - 56s 594ms/step - loss: 0.2649 - accuracy: 0.9060 - val_loss: 0.2925 - val_accuracy:
Epoch 11/50
94/94 [==============================] - 56s 595ms/step - loss: 0.2616 - accuracy: 0.9087 - val_loss: 0.3087 - val_accuracy:
Epoch 12/50
94/94 [==============================] - 56s 597ms/step - loss: 0.2446 - accuracy: 0.9131 - val_loss: 0.2944 - val_accuracy:
Epoch 13/50
94/94 [==============================] - 56s 594ms/step - loss: 0.2361 - accuracy: 0.9153 - val_loss: 0.3020 - val_accuracy:
Epoch 14/50
94/94 [==============================] - 56s 595ms/step - loss: 0.2271 - accuracy: 0.9181 - val_loss: 0.3106 - val_accuracy:
Epoch 15/50
94/94 [==============================] - 56s 594ms/step - loss: 0.2171 - accuracy: 0.9211 - val_loss: 0.2981 - val_accuracy:
Epoch 16/50
94/94 [==============================] - 56s 594ms/step - loss: 0.2043 - accuracy: 0.9270 - val_loss: 0.2990 - val_accuracy:
Epoch 17/50
94/94 [==============================] - 56s 594ms/step - loss: 0.2003 - accuracy: 0.9285 - val_loss: 0.2841 - val_accuracy:
Epoch 18/50
94/94 [==============================] - 56s 601ms/step - loss: 0.1916 - accuracy: 0.9311 - val_loss: 0.3000 - val_accuracy:
Epoch 19/50
94/94 [==============================] - 56s 593ms/step - loss: 0.1853 - accuracy: 0.9342 - val_loss: 0.2966 - val_accuracy:
Epoch 20/50
94/94 [==============================] - 56s 591ms/step - loss: 0.1768 - accuracy: 0.9365 - val_loss: 0.3003 - val_accuracy:
Epoch 21/50
94/94 [==============================] - 55s 587ms/step - loss: 0.1679 - accuracy: 0.9395 - val_loss: 0.3320 - val_accuracy:
Epoch 22/50
94/94 [==============================] - 55s 587ms/step - loss: 0.1663 - accuracy: 0.9400 - val_loss: 0.3158 - val_accuracy:
Epoch 23/50
94/94 [==============================] - 55s 588ms/step - loss: 0.1602 - accuracy: 0.9427 - val_loss: 0.3072 - val_accuracy:
Epoch 24/50
94/94 [==============================] - 55s 586ms/step - loss: 0.1559 - accuracy: 0.9424 - val_loss: 0.3272 - val_accuracy:
Epoch 25/50
94/94 [==============================] - 55s 586ms/step - loss: 0.1516 - accuracy: 0.9447 - val_loss: 0.3292 - val_accuracy:
Epoch 26/50
94/94 [==============================] - 55s 584ms/step - loss: 0.1499 - accuracy: 0.9462 - val_loss: 0.3459 - val_accuracy:
Epoch 27/50
94/94 [==============================] - 55s 585ms/step - loss: 0.1391 - accuracy: 0.9498 - val_loss: 0.3359 - val_accuracy:
Epoch 28/50
94/94 [==============================] - 55s 585ms/step - loss: 0.1348 - accuracy: 0.9516 - val_loss: 0.3397 - val_accuracy:
Epoch 29/50
```

CODE

The code of my project can be accessed from the below link:

https://colab.research.google.com/drive/1Dlrhz3-OCCwCFSo99GuvCtTNQNl8tPl7?usp=sharing

CONCLUSION

- In this project, I was able to build fashion apparel recognition using a Convolutional Neural Network (CNN).
- I was able to train and test the model for making its predictions.
- I also created 2 complex CNN for comparison of their accuracies.
- **The project consists of:**

I. A Fashion MNIST Classification model made using Convolutional neural networks(CNN).

II. 2 complex CNN models.

FIRST CNN MODEL

2 COMPLEX CNN

ACCURACY: 91%

```
# Training model
model_2.fit(X_train, y_train, epochs = 20, batch_size = 512, verbose = 1, validation_data = (X_val, y_val))

    Epoch 1/20
    94/94 [==============================] - 28s 280ms/step - loss: 2.0244 - accuracy: 0.5850 - val_loss: 0.5536 - val_accuracy: 0.79
    Epoch 2/20
    94/94 [==============================] - 27s 291ms/step - loss: 0.5929 - accuracy: 0.7799 - val_loss: 0.4583 - val_accuracy: 0.83
    Epoch 3/20
    94/94 [==============================] - 34s 361ms/step - loss: 0.4972 - accuracy: 0.8182 - val_loss: 0.4096 - val_accuracy: 0.84
    Epoch 4/20
    94/94 [==============================] - 26s 278ms/step - loss: 0.4455 - accuracy: 0.8365 - val_loss: 0.3908 - val_accuracy: 0.85
    Epoch 5/20
    94/94 [==============================] - 26s 276ms/step - loss: 0.4086 - accuracy: 0.8485 - val_loss: 0.3901 - val_accuracy: 0.85
    Epoch 6/20
    94/94 [==============================] - 26s 275ms/step - loss: 0.3891 - accuracy: 0.8569 - val_loss: 0.3651 - val_accuracy: 0.86
    Epoch 7/20
    94/94 [==============================] - 26s 277ms/step - loss: 0.3628 - accuracy: 0.8672 - val_loss: 0.3491 - val_accuracy: 0.87
    Epoch 8/20
    94/94 [==============================] - 27s 290ms/step - loss: 0.3466 - accuracy: 0.8720 - val_loss: 0.3381 - val_accuracy: 0.87
    Epoch 9/20
    94/94 [==============================] - 27s 291ms/step - loss: 0.3273 - accuracy: 0.8780 - val_loss: 0.3395 - val_accuracy: 0.87
    Epoch 10/20
    94/94 [==============================] - 27s 283ms/step - loss: 0.3174 - accuracy: 0.8823 - val_loss: 0.3375 - val_accuracy: 0.87
    Epoch 11/20
    94/94 [==============================] - 26s 277ms/step - loss: 0.3030 - accuracy: 0.8867 - val_loss: 0.3328 - val_accuracy: 0.87
    Epoch 12/20
    94/94 [==============================] - 26s 276ms/step - loss: 0.2928 - accuracy: 0.8896 - val_loss: 0.3185 - val_accuracy: 0.88
    Epoch 13/20
    94/94 [==============================] - 26s 278ms/step - loss: 0.2803 - accuracy: 0.8943 - val_loss: 0.3081 - val_accuracy: 0.88
    Epoch 14/20
    94/94 [==============================] - 26s 277ms/step - loss: 0.2743 - accuracy: 0.8991 - val_loss: 0.3172 - val_accuracy: 0.88
    Epoch 15/20
    94/94 [==============================] - 26s 280ms/step - loss: 0.2653 - accuracy: 0.9009 - val_loss: 0.3123 - val_accuracy: 0.88
    Epoch 16/20
    94/94 [==============================] - 34s 358ms/step - loss: 0.2573 - accuracy: 0.9039 - val_loss: 0.3041 - val_accuracy: 0.89
    Epoch 17/20
    94/94 [==============================] - 40s 431ms/step - loss: 0.2462 - accuracy: 0.9063 - val_loss: 0.3037 - val_accuracy: 0.89
    Epoch 18/20
    94/94 [==============================] - 34s 361ms/step - loss: 0.2480 - accuracy: 0.9071 - val_loss: 0.3083 - val_accuracy: 0.89
    Epoch 19/20
    94/94 [==============================] - 27s 289ms/step - loss: 0.2394 - accuracy: 0.9094 - val_loss: 0.3125 - val_accuracy: 0.89
    Epoch 20/20
    94/94 [==============================] - 30s 315ms/step - loss: 0.2274 - accuracy: 0.9141 - val_loss: 0.2999 - val_accuracy: 0.89
    <keras.callbacks.History at 0x7f923b5c8f90>

model_2.evaluate(X_test, y_test)
```

ACCURACY: 96%(midst) & 92%(final)

```
94/94 [==============================] - 56s 598ms/step - loss: 0.3138 - accuracy: 0.8885 - val_loss: 0.3400 - val_accuracy:
Epoch 8/50
94/94 [==============================] - 56s 598ms/step - loss: 0.2968 - accuracy: 0.8951 - val_loss: 0.3173 - val_accuracy:
Epoch 9/50
94/94 [==============================] - 56s 598ms/step - loss: 0.2776 - accuracy: 0.9014 - val_loss: 0.3064 - val_accuracy:
Epoch 10/50
94/94 [==============================] - 56s 594ms/step - loss: 0.2649 - accuracy: 0.9060 - val_loss: 0.2925 - val_accuracy:
Epoch 11/50
94/94 [==============================] - 56s 595ms/step - loss: 0.2616 - accuracy: 0.9087 - val_loss: 0.3087 - val_accuracy:
Epoch 12/50
94/94 [==============================] - 56s 597ms/step - loss: 0.2446 - accuracy: 0.9131 - val_loss: 0.2944 - val_accuracy:
Epoch 13/50
94/94 [==============================] - 56s 594ms/step - loss: 0.2361 - accuracy: 0.9153 - val_loss: 0.3020 - val_accuracy:
Epoch 14/50
94/94 [==============================] - 56s 595ms/step - loss: 0.2271 - accuracy: 0.9181 - val_loss: 0.3106 - val_accuracy:
Epoch 15/50
94/94 [==============================] - 56s 594ms/step - loss: 0.2171 - accuracy: 0.9211 - val_loss: 0.2981 - val_accuracy:
Epoch 16/50
94/94 [==============================] - 56s 594ms/step - loss: 0.2043 - accuracy: 0.9270 - val_loss: 0.2990 - val_accuracy:
Epoch 17/50
94/94 [==============================] - 56s 594ms/step - loss: 0.2003 - accuracy: 0.9285 - val_loss: 0.2841 - val_accuracy:
Epoch 18/50
94/94 [==============================] - 56s 601ms/step - loss: 0.1916 - accuracy: 0.9311 - val_loss: 0.3000 - val_accuracy:
Epoch 19/50
94/94 [==============================] - 56s 593ms/step - loss: 0.1853 - accuracy: 0.9342 - val_loss: 0.2966 - val_accuracy:
Epoch 20/50
94/94 [==============================] - 56s 591ms/step - loss: 0.1768 - accuracy: 0.9365 - val_loss: 0.3003 - val_accuracy:
Epoch 21/50
94/94 [==============================] - 55s 587ms/step - loss: 0.1679 - accuracy: 0.9395 - val_loss: 0.3320 - val_accuracy:
Epoch 22/50
94/94 [==============================] - 55s 587ms/step - loss: 0.1663 - accuracy: 0.9400 - val_loss: 0.3158 - val_accuracy:
Epoch 23/50
94/94 [==============================] - 55s 588ms/step - loss: 0.1602 - accuracy: 0.9427 - val_loss: 0.3072 - val_accuracy:
Epoch 24/50
94/94 [==============================] - 55s 586ms/step - loss: 0.1559 - accuracy: 0.9424 - val_loss: 0.3272 - val_accuracy:
Epoch 25/50
94/94 [==============================] - 55s 586ms/step - loss: 0.1516 - accuracy: 0.9447 - val_loss: 0.3292 - val_accuracy:
Epoch 26/50
94/94 [==============================] - 55s 584ms/step - loss: 0.1499 - accuracy: 0.9462 - val_loss: 0.3459 - val_accuracy:
Epoch 27/50
94/94 [==============================] - 55s 585ms/step - loss: 0.1391 - accuracy: 0.9498 - val_loss: 0.3359 - val_accuracy:
Epoch 28/50
94/94 [==============================] - 55s 585ms/step - loss: 0.1348 - accuracy: 0.9516 - val_loss: 0.3397 - val_accuracy:
Epoch 29/50
```

THE END

THANK YOU!