

# **Image Classification Using Convolutional Neural Network**

# **INDEX**

1. Introduction
2. Problem Statement
3. Objectives
4. Dataset Description
5. Tools & Technologies Used
6. System Architecture
7. Model Design & Implementation
8. Training Process
9. Model Saving & Label Encoding
10. Application Development (Streamlit)
11. Prediction Workflow
12. Results & Observations
13. Challenges Faced
14. Best Practices Followed
15. Conclusion

# Image Classification Using CNN

## 1. Introduction

Fashion image classification plays an important role in e-commerce platforms, enabling automatic tagging, recommendation systems, and visual search. This project implements a Convolutional Neural Network (CNN) to classify fashion images into different clothing categories and deploys the trained model using a Streamlit web application.

## 2. Problem Statement

Manual classification of fashion products is time-consuming and error-prone. The goal is to automatically classify fashion images into their respective categories using deep learning techniques.

## 3. Objectives

- To build a CNN-based image classification model
- To train the model on a real-world fashion dataset
- To save the trained model and class labels
- To develop a user-friendly web interface using Streamlit
- To predict fashion categories from uploaded images

## 4. Dataset Description

The dataset used is the Myntra Fashion Dataset, consisting of:

- Fashion product images (`.jpg`)
- Metadata stored in `styles.csv`

Dataset: <https://www.kaggle.com/datasets/paramagarwal/fashion-product-images-small>

## 5. Tools & Technologies Used

<b>Component</b>	<b>Technology</b>
<b>Programming Language</b>	Python 3.10
<b>Deep Learning</b>	TensorFlow/Keras
<b>Image Processing</b>	PIL, OpenCV
<b>Data Handling</b>	NumPy, Pandas
<b>Model Training</b>	CNN
<b>Label Encoding</b>	Scikit-learn
<b>Web Interface</b>	Streamlit

## 6. System Architecture

Image Dataset → Preprocessing → CNN Model → Model Training



Saved Model + Labels



Streamlit Application



Prediction Output

## 7. Model Design & Implementation

The CNN model is built using **Sequential** API:

- Conv2D (32 filters)
- MaxPooling
- Conv2D (64 filters)
- MaxPooling
- Conv2D (128 filters)
- MaxPooling
- Flatten
- Dense (128 units)
- Output Dense layer (Softmax)

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(len(label_encoder.classes_), activation='softmax'))

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

## 8. Training Process

### Data Preparation

- Image resizing to 150×150
- Normalization using `rescale=1./255`
- Data augmentation:
  - Horizontal flip
  - Zoom
  - Shear

### Data Splitting

- 80% Training
- 20% Validation
- 20% Testing (from training data)

## Training

```
history = model.fit(  
    train_generator,  
    epochs=10,  
    validation_data=validation_generator  
)
```

## 9. Model Saving & Label Encoding

```
# Save the model  
model.save("model.h5")  
  
# Save the label encoder classes  
np.save("label.npy", label_encoder.classes_)
```

## 10. Application Development (Streamlit)

The web application allows users to:

- Upload an image
- View the uploaded image
- Receive predicted fashion category

### Features:

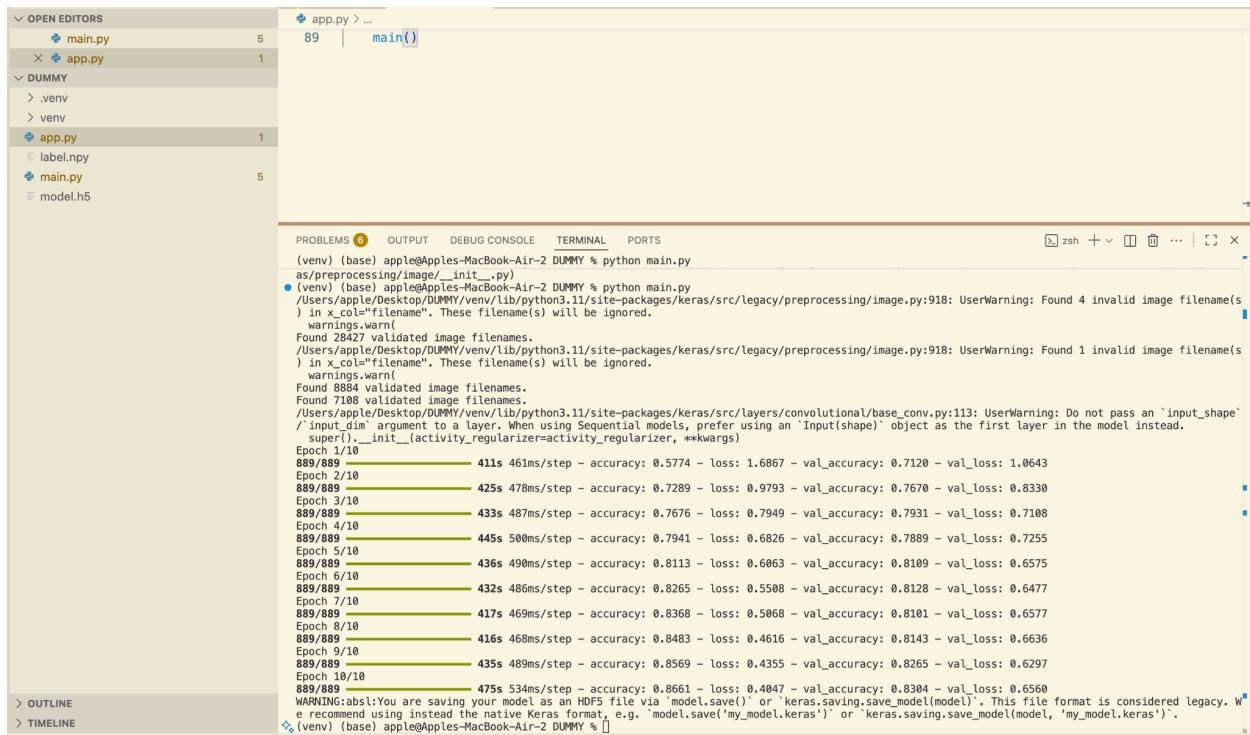
- Custom background image
- Styled UI
- Real-time prediction

## 11. Prediction Workflow

1. User uploads an image
2. Image is resized to **150×150**
3. Pixel values normalized
4. Image passed to CNN model
5. Softmax output generated
6. Highest probability class selected
7. Label decoded using LabelEncoder

## 12. Results & Observations

- Model successfully predicts fashion categories
- Accurate classification for clear clothing images
- Works well for common apparel types
- Performance depends on image quality



The screenshot shows a code editor interface with several files open in the left sidebar:

- OPEN EDITORS:**
  - main.py (5)
  - app.py (1)
  - DUMMY
  - .venv
  - venv
  - app.py (1)
  - label.npy
  - main.py (5)
  - model.h5

The main area displays the content of the **app.py** file:

```

app.py > ...
89 |     main()

```

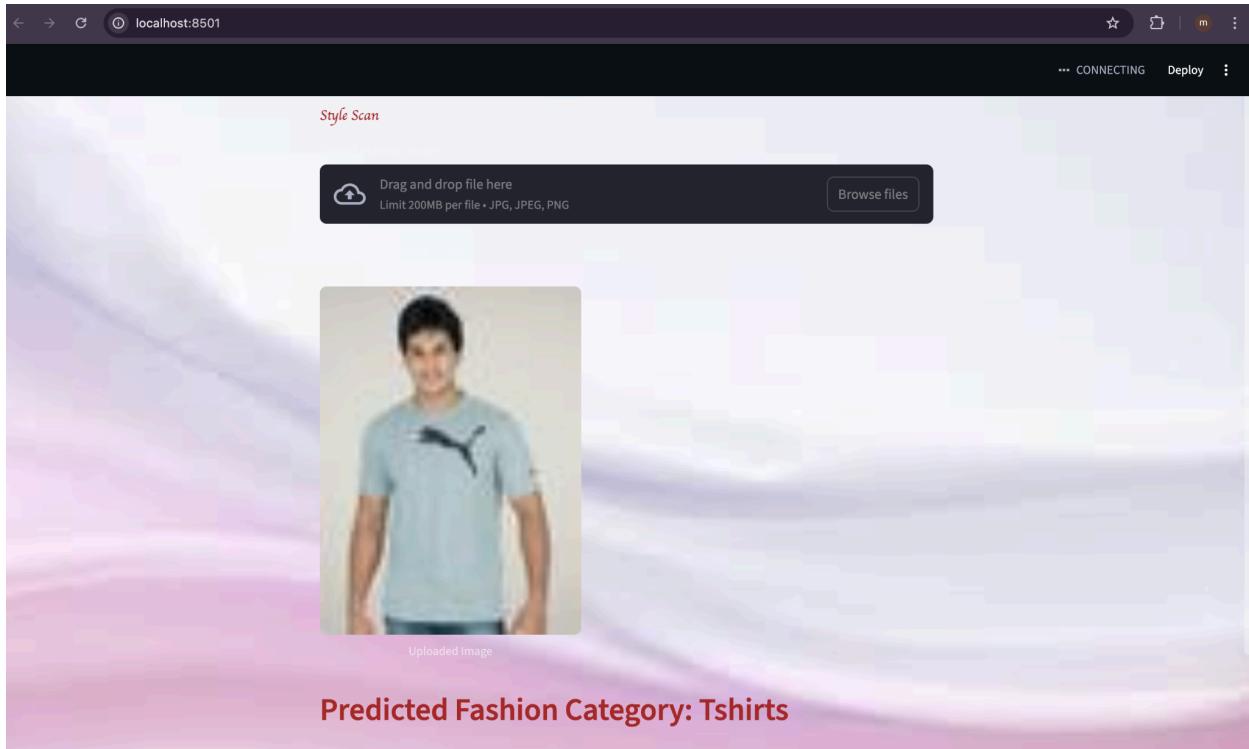
Below the code editor is the **PROBLEMS** tab of the terminal interface, showing the following log output:

```

(venv) (base) apple@Apple-MacBook-Air-2: DUMMY % python main.py
as/preprocessing/image/_init_.py)
(venv) (base) apple@Apple-MacBook-Air-2: DUMMY % python main.py
/Users/apple/Desktop/DUMMY/venv/lib/python3.11/site-packages/keras/src/legacy/preprocessing/image.py:918: UserWarning: Found 4 invalid image filename(s) in x_col="filename". These filename(s) will be ignored.
  warnings.warn(
Found 28427 validated image filenames.
/Users/apple/Desktop/DUMMY/venv/lib/python3.11/site-packages/keras/src/legacy/preprocessing/image.py:918: UserWarning: Found 1 invalid image filename(s) in x_col="filename". These filename(s) will be ignored.
  warnings.warn(
Found 8884 validated image filenames.
Found 7108 validated image filenames.
/Users/apple/Desktop/DUMMY/venv/lib/python3.11/site-packages/keras/src/layers/convolutional/base_conv.py:113: UserWarning: Do not pass an `input_shape` /`input` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/10
889/889 411s 461ms/step - accuracy: 0.5774 - loss: 1.6867 - val_accuracy: 0.7120 - val_loss: 1.0643
Epoch 2/10
889/889 425s 478ms/step - accuracy: 0.7289 - loss: 0.9793 - val_accuracy: 0.7670 - val_loss: 0.8330
Epoch 3/10
889/889 433s 487ms/step - accuracy: 0.7676 - loss: 0.7949 - val_accuracy: 0.7931 - val_loss: 0.7108
Epoch 4/10
889/889 445s 500ms/step - accuracy: 0.7941 - loss: 0.6826 - val_accuracy: 0.7889 - val_loss: 0.7255
Epoch 5/10
889/889 436s 490ms/step - accuracy: 0.8113 - loss: 0.6063 - val_accuracy: 0.8109 - val_loss: 0.6575
Epoch 6/10
889/889 432s 486ms/step - accuracy: 0.8265 - loss: 0.5508 - val_accuracy: 0.8128 - val_loss: 0.6477
Epoch 7/10
889/889 417s 469ms/step - accuracy: 0.8368 - loss: 0.5068 - val_accuracy: 0.8101 - val_loss: 0.6577
Epoch 8/10
889/889 416s 468ms/step - accuracy: 0.8483 - loss: 0.4616 - val_accuracy: 0.8143 - val_loss: 0.6636
Epoch 9/10
889/889 435s 489ms/step - accuracy: 0.8569 - loss: 0.4355 - val_accuracy: 0.8265 - val_loss: 0.6297
Epoch 10/10
889/889 475s 534ms/step - accuracy: 0.8661 - loss: 0.4047 - val_accuracy: 0.8304 - val_loss: 0.6560
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.

```

The bottom of the terminal shows the command: `(venv) (base) apple@Apple-MacBook-Air-2: DUMMY %`.



## 13. Challenges Faced

- TensorFlow & Keras version incompatibility
- .h5 model deserialization issues
- Python 3.11 compatibility problems
- Environment dependency conflicts

## 14. Best Practices Followed

- Data augmentation to reduce overfitting
- Label encoding consistency
- Separate training and inference scripts
- Modular preprocessing functions
- Clean UI with Streamlit

## 15. Conclusion

This project successfully demonstrates how **convolutional neural networks(CNNs)** can be used to classify fashion images effectively. By integrating TensorFlow with Streamlit, the system provides an intuitive and interactive user experience. The project highlights the importance of proper environment management and model format compatibility.