## Images reading and processing

In [1]:
```
import cv2
import os
import matplotlib.pyplot as plt
import numpy as np
```

In [2]:
```
# to read images from each file and assigning it a label
def load_images_from_folder(folder):
    images = []
    lables=[]
    c=-1
    d={}
    for file in os.listdir(folder):
        c+=1
        d[file]=c
        for filename in os.listdir(os.path.join(folder,file)):
            img = cv2.imread(os.path.join(folder,file,filename))
            if img is not None:
                images.append(img)
                lables.append([c])
    return images,lables
```

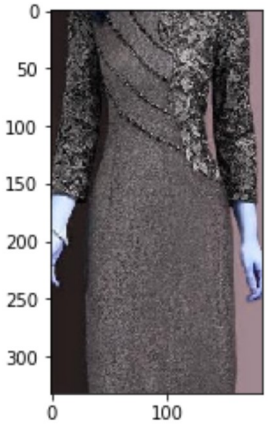
In [3]:
```
folder='dataset'

images,lables=load_images_from_folder(folder)
```

In [4]:
```
#checking the length of the data read
print(len(images))
print(len(lables))
```

```
2007
2007
```

In [5]:
```
#checking for the images
plt.imshow(images[0])
```

Out[5]:
```
<matplotlib.image.AxesImage at 0x29380572358>
```

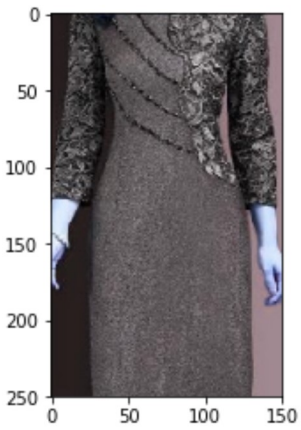


In [6]:
```
#images shape
images[0].shape
```

Out[6]: (332, 183, 3)

In [7]:
```
img = cv2.resize(images[0],(150,250))

plt.imshow(img)
```

Out[7]: <matplotlib.image.AxesImage at 0x29383b473c8>



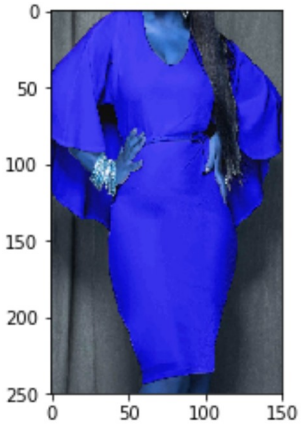

As every image is of different size we will resize them to a feasible size and make the size of each image similar.

In [8]:
```
#resizeing the images
cleaned_images=[]
for i in range(len(images)):
    #image = cv2.cvtColor(images[i], cv2.COLOR_RGB2GRAY)
    #image = cv2.cvtColor(images[i], cv2.COLOR_BGR2GRAY)
    cleaned_images.append(cv2.resize(images[i],(150,250)))
```

In [9]:
```
#splitting the data into two parts that is test and train with train having 20% of
the total data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(cleaned_images, lables, test_si
ze=0.2, random_state=42)
```

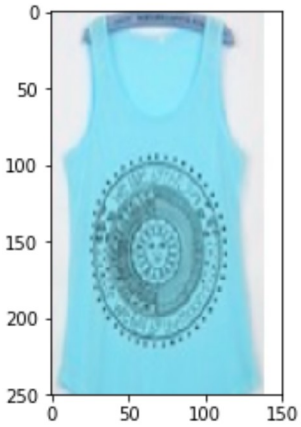
In [10]:
```
#checking if every this is right upto this
plt.imshow(X_train[0])
```

Out[10]: <matplotlib.image.AxesImage at 0x293ac765ac8>



In [11]:
```
plt.imshow(X_test[0])
```

Out[11]:
```
<matplotlib.image.AxesImage at 0x293b06ac358>
```




In [12]:
```
print(type(X_test))
print(type(X_train))
```

```
<class 'list'>
<class 'list'>
```

In [13]:
```
#converting them to numpy array type so that it may be further used to as type conv
ersions
X_test=np.array(X_test)
X_train=np.array(X_train)

X_train = X_train.astype('float32')/255
X_test = X_test.astype('float32')/255
```

In [14]:
```
print(X_train.shape)

print(X_test.shape)
```

```
(1605, 250, 150, 3)
(402, 250, 150, 3)
```

so far so good

In [15]:
```
from keras.utils import np_utils
import keras

# one-hot encode the labels
num_classes = len(np.unique(y_train))
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

# break training set into training and validation sets
(X_train, X_valid) = X_train[100:], X_train[:100]
(y_train, y_valid) = y_train[100:], y_train[:100]

# print shape of training set
print('x_train shape:', X_train.shape)

# print number of training, validation, and test images
print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')
print(X_valid.shape[0], 'validation samples')
```

```
C:\Users\Akarsh Somani\Anaconda3\lib\site-packages\h5py\__init__.py:36: FutureWa
rning: Conversion of the second argument of issubdtype from `float` to `np.float
ing` is deprecated. In future, it will be treated as `np.float64 == np.dtype(flo
at).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

```
x_train shape: (1505, 250, 150, 3)
1505 train samples
402 test samples
100 validation samples
```

In [16]:
```
#Our CNN model
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

model = Sequential()
model.add(Conv2D(filters=16, kernel_size=3, padding='same', activation='relu',
                 input_shape=(250, 150, 3)))
model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(filters=32, kernel_size=2, padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.2))
model.add(Conv2D(filters=64, kernel_size=2, padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(500, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(13, activation='softmax'))

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 250, 150, 16)      448
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 125, 75, 16)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 125, 75, 32)       2080
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 62, 37, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 62, 37, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 62, 37, 64)        8256
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 31, 18, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 31, 18, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 35712)             0
_________________________________________________________________
dense_1 (Dense)              (None, 500)               17856500
_________________________________________________________________
dropout_3 (Dropout)          (None, 500)               0
_________________________________________________________________
dense_2 (Dense)              (None, 13)                6513
=================================================================
Total params: 17,873,797
Trainable params: 17,873,797
Non-trainable params: 0
_________________________________________________________________
```

In [17]:
```
# compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam',
                  metrics=['accuracy'])
```

In [18]:
```
from keras.callbacks import ModelCheckpoint

# train the model
checkpointer = ModelCheckpoint(filepath='model.weights.best.hdf5', verbose=1,
                               save_best_only=True)
hist = model.fit(X_train, y_train, batch_size=100, epochs=20,
          validation_data=(X_valid, y_valid), callbacks=[checkpointer],shuffle=True
)
```

```
Train on 1505 samples, validate on 100 samples
Epoch 1/20
1505/1505 [==============================] - 12s 8ms/step - loss: 5.3094 - acc:
0.0804 - val_loss: 2.5123 - val_acc: 0.1400

Epoch 00001: val_loss improved from inf to 2.51231, saving model to model.weight
s.best.hdf5
Epoch 2/20
1505/1505 [==============================] - 7s 5ms/step - loss: 2.5759 - acc: 0
.0817 - val_loss: 2.5599 - val_acc: 0.1700

Epoch 00002: val_loss did not improve from 2.51231
Epoch 3/20
1505/1505 [==============================] - 7s 5ms/step - loss: 2.5580 - acc: 0
.1110 - val_loss: 2.5603 - val_acc: 0.0800

Epoch 00003: val_loss did not improve from 2.51231
Epoch 4/20
1505/1505 [==============================] - 7s 5ms/step - loss: 2.5388 - acc: 0
.1236 - val_loss: 2.5308 - val_acc: 0.1500

Epoch 00004: val_loss did not improve from 2.51231
Epoch 5/20
1505/1505 [==============================] - 7s 5ms/step - loss: 2.4933 - acc: 0
.1548 - val_loss: 2.5004 - val_acc: 0.0900

Epoch 00005: val_loss improved from 2.51231 to 2.50039, saving model to model.we
ights.best.hdf5
Epoch 6/20
1505/1505 [==============================] - 7s 5ms/step - loss: 2.4038 - acc: 0
.2053 - val_loss: 2.4175 - val_acc: 0.2200

Epoch 00006: val_loss improved from 2.50039 to 2.41755, saving model to model.we
ights.best.hdf5
Epoch 7/20
1505/1505 [==============================] - 7s 5ms/step - loss: 2.3000 - acc: 0
.2319 - val_loss: 2.4726 - val_acc: 0.1600

Epoch 00007: val_loss did not improve from 2.41755
Epoch 8/20
1505/1505 [==============================] - 7s 5ms/step - loss: 2.2102 - acc: 0
.2757 - val_loss: 2.3937 - val_acc: 0.2100

Epoch 00008: val_loss improved from 2.41755 to 2.39367, saving model to model.we
ights.best.hdf5
Epoch 9/20
1505/1505 [==============================] - 7s 5ms/step - loss: 2.0681 - acc: 0
.3203 - val_loss: 2.3915 - val_acc: 0.2300

Epoch 00009: val_loss improved from 2.39367 to 2.39149, saving model to model.we
ights.best.hdf5
Epoch 10/20
1505/1505 [==============================] - 7s 5ms/step - loss: 1.8587 - acc: 0
.3967 - val_loss: 2.3957 - val_acc: 0.2400

Epoch 00010: val_loss did not improve from 2.39149
Epoch 11/20
1505/1505 [==============================] - 7s 5ms/step - loss: 1.6446 - acc: 0
.4950 - val_loss: 2.3905 - val_acc: 0.2500

Epoch 00011: val_loss improved from 2.39149 to 2.39050, saving model to model.we
ights.best.hdf5
Epoch 12/20
1505/1505 [==============================] - 7s 5ms/step - loss: 1.4765 - acc: 0
```

In [19]:
```
# load the weights that yielded the best validation accuracy
model.load_weights('model.weights.best.hdf5')
```

In [20]:
```
# evaluate and print test accuracy
score = model.evaluate(X_test, y_test, verbose=0)
print('\n', 'Test accuracy:', score[1])
```

```
 Test accuracy: 0.24378109474976858
```