Fashion Image Classifier using Machine Learning

Tracy L. Keys, *Graduate Student University of Southern California*

Abstract This report describes my project to build a convolutional neural net to classify fashion images similar to Fashion MNIST, utilizing transfer learning from ImageNet.

***Index Terms*—Multiclass Classification, CNN, Fashion MNIST, Machine Learning, VGG-16, Resnet 50, Deep learning, transfer learning, ImageNet**

I. INTRODUCTION

Those who shop for fashion online know the frustration of searching and trawling through multiple sites looking for something in particular, and when you finally do find it, it's out of stock in your size, and you must start all over again.

I dream of one day selling my search plug-in to Google to find and curate clothing from online sites that are in stock, are the right size, are in my budget, and all the other factors that I'm searching for.

To enable this, I would build a tool that uses search terms, and/or an image or a description of the item and it will search the web for me.

This project is a prototype to see how one would go about doing this, and whether machine learning makes it at all feasible.

Focusing on the image recognition aspect of the problem, I have built my own fashion data set from searching the internet and built and tuned machine learning models (convolutional neural networks (CNNs)) to see which works best for finding the images I am searching for.

II. BACKGROUND

My proposal comes about from a desire to solve a personal pain point, as I am a prolific online shopper. I've recently been encouraged by Google's own product development for Google Search. Whenever people perform searches regularly, Google eventually brings out a specific tool for each kind of search, such as directions in Google Maps, and more recently, the ability to search airlines and book flights and hotels. I hope that this enhanced Fashion Search tool is just around the corner, but in the meantime, I will build my own.

III. OBJECTIVES

The research question for this paper is "what is the best performing Machine Learning solution to accurately classify fashion images?"

The two primary deliverables of this project are:

- Creation of a labelled data set for use in my model,
- An evaluation of machine learning and deep learning models for Fashion Image classification,

Being a team of one, my instructions for this project as outlined in class by Professor Muslea is to apply 3-5 machine learning algorithm to my dataset, and then experiment to improve the out-of-the-box results.

IV. METHODOLOGY

Due to the availability of online tutorials and documentation, I chose to use Keras with a Tensorflow back end, using Python language to build my data set and models.

The midterm objective was to build the initial small data set and train and evaluate two machine learning models end to end, which I accomplished, and whose methodology and results will be outlined below and in Section V.

The objective of the final paper was to expand the data set to ten classes like Fashion MNIST [1], develop more models, and improve the accuracy of the models, with the benchmark for performance being estimated human accuracy of 95%. Since the initial plan, I decided rather than spend time on routine work such as expanding my dataset to 10 classes, I have instead focused on transfer learning: fine tuning the VGG16 [2] model and the deeper CNN Resnet50[3] to gain practical experience engineering deep learning models.

A. Dataset

1) Creation of the dataset

The creator of Keras, Francois Chollet [4] outlined in the Keras blog an image classification CNN with over 94% accuracy on as little as 1000 images per class. Therefore, my objective was to obtain a minimum of 1000 images per class for my data set.

Initially, I scraped 100 images for each of three classes: Dresses, Pullovers and Shirts. Unfortunately, the current method I am using has a limit of 100 images [5] per search term.
To bring the data set up to 1,000 images per class, I specified the colors for each search i.e. red dress, blue dress, yellow dress and so on, to work around the limit. The search term was the folder the images were placed in, and once arranged into the 3 classes (dresses, shirts and pullovers), become the class labels.

2) Data pre-processing

The dataset required cleaning as some images were unreadable. Then I utilized data augmentation using Keras Image Data Generation [6] to change the images to bring the total images per class to 1000. If required, in future I could perform web scraping using Selenium web driver [5], or try using Bing Image API to more quickly increase the size of the dataset, which doesn't have this limitation.

Keras Image Data Generation [6] takes each image and distorts it to create slightly different versions that are still useful for training the machine learning algorithms.

The Keras GitHub page [7] has code to augment the images for the cats and dogs Kaggle dataset, which I have adapted for my data set as show in Figure 1 below.

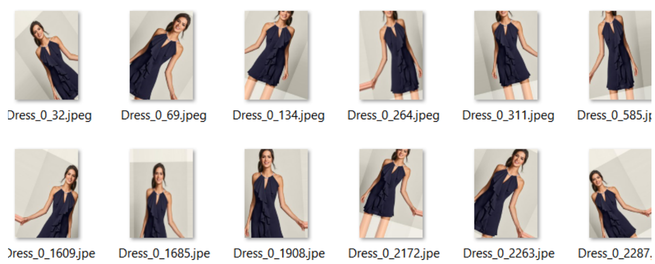

Fig. 1. Data augmentation using Keras image data generator tool on my dataset

I used Keras flow.from.data function to enable preprocessing these 224 x 224 images into the 255-pixel scale. This function can also augment the images in multiple other ways, such as rotating or shifting the image to enable training on more images even though the dataset is small. After the midterm, I also changed the shape of the image dataset from a 3D to a 2D array to give me access to other code templates for calculating test loss and accuracy, which I was struggling to do in some cases when completing my midterm paper [8].

The other dataset I used is ImageNet [9], [10] indirectly, because both VGG[11] and Resnet 50 [12]are pre-trained on ImageNet.[9], [10]. ImageNet has 1000 classes of images, including items of apparel and at least 1000 images per class.

3) Dataset split

In order to ensure the accuracy of the

measurements of model performance, I performed training and validation using two different splits of my dataset. 20% (600) of the images were held back as the test set in both cases. For the remaining 80% of data, I split the training and validation sets 80/20 for the initial VGG16 model, the tuned VGG16 model and the Resnet50 model (outlined in Part B below).

Dietterich [13] recommends splitting training and validation data 50/50, therefore I also ran the VGG16 model (which was the best performing, as will be explained in Section V) using the 50/50 split recommended. This ensures no overlap between the training and validation data because in the first run, 50% is training data, then that same 50% is used as validation data in the second run.

4) Limitations of dataset

The dataset is just three classes: dress, pullover and shirt. These items are quite similar, and there is some mislabeling within the dataset. This has been accommodated within the allowance for 5% error rate.

B. Models

My research question requires the use of a multi-class classification model, and therefore there are certain functions that are useful in this case.

At the time of the mid-term paper draft deadline, I had implemented a basic CNN [14] and also a VGG-16 pre-trained model [11] as shown in Figure1. This was based on code from deeplizard on YouTube [15]. I applied transfer learning from the weights learned by this model on ImageNet data to my Fashion dataset.

Each hidden layer improves the generalizability of the model, and therefore should improve the accuracy on the test set.

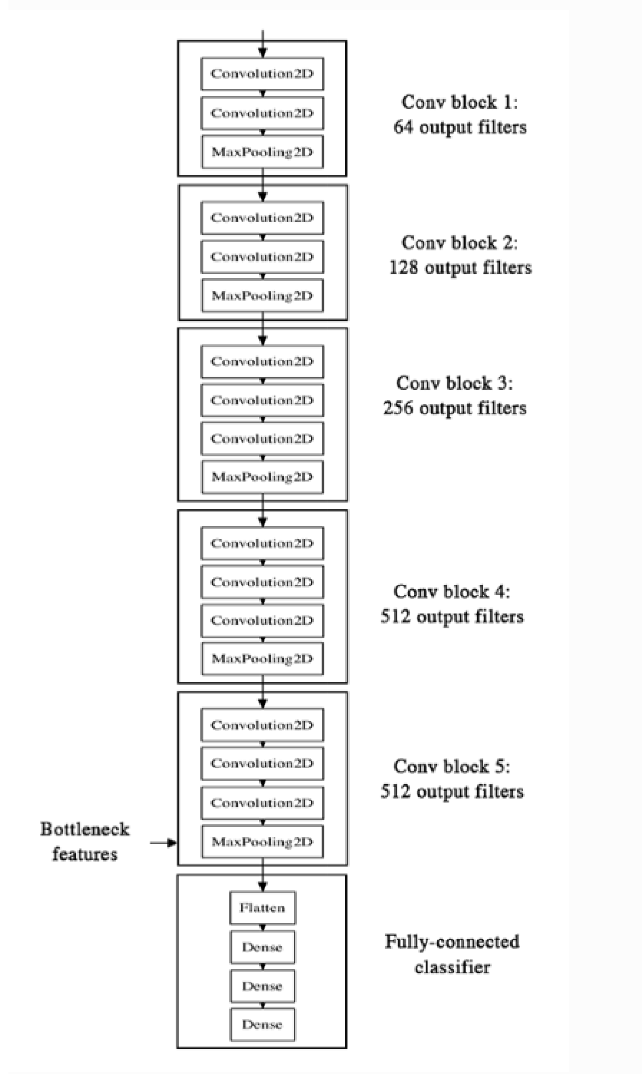

Figure 1Visual Geometry Group VGG16 CNN

After completing the midterm, the results indicated that there was too much bias in my model. Therefore, I took two courses of action to improve the performance. Firstly, I decided to tune the hyperparameters of the VGG16 model, and secondly trial a deeper Resnet50 model [12] with 50 rather than 16 hidden layers (also with pre-trained weights on the ImageNet dataset). These two models were adapted from the OpenCV website and code provided by Mallick [8].

In order to fine tune the models, I applied dropout to the convolutional layers, and changed the learning rate, and as shown in Figure 4 this improved the accuracy significantly. [16]

Resnet50 is a CNN with many more layers than

VGG16, however it deals with the vanishing gradient problem that comes from deep layers by applying the identity matrix to allow the gradient to be passed through each convolution [12].

C. *Performance Metrics*

In order to benchmark model performance, human accuracy is estimated to be 95%. 100% isn't likely, as the class of some items may be debatable (remember the blue/black vs white/gold dress internet craze?), and there is some mislabeling in the dataset.

In this project, machine learning performance is measured twice.

Firstly, the performance of the model after learning on the training set is measured on the validation set, and the metric used is validation loss (categorical cross entropy) and accuracy. The model is trained over 20 epochs twice. The second time performance is measured is on the unseen test set, and the metric is categorical cross entropy loss and accuracy.

In order to draw conclusions about the accuracy of my model on unseen data in future, I calculated the accuracy range at 95% confidence using t-scores, because the accuracy rate of the entire population is not known [17].

V. RESULTS

1) *Midterm results*

Parameters and results for the two models I evaluated for the midterm are shown in Figure 4. I had adapted the code for these two models from deeplizard[15]. Through changing the learning rate for the Basic CNN from 0.001 to 0.01, validation accuracy performance improved from basically worse than chance (25%) to chance 33%. But then it did not change over the epochs, as shown in Figure 2. The same result was visible when I increased the training and validation epochs to 20.

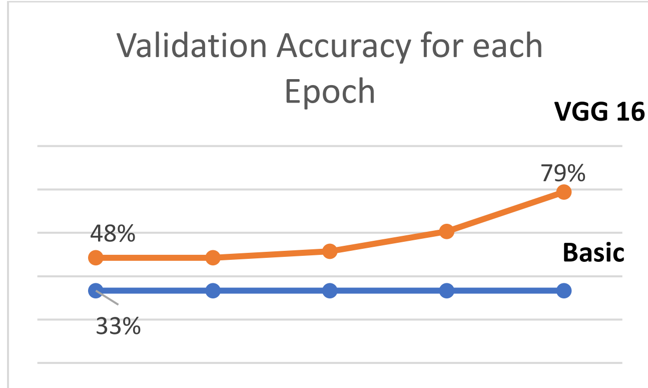

Figure 2Midterm results

The basic CNN is essentially predicting the same class every time, bias is very high and therefore the accuracy is very low, as shown in the confusion matrix in Figure 3.

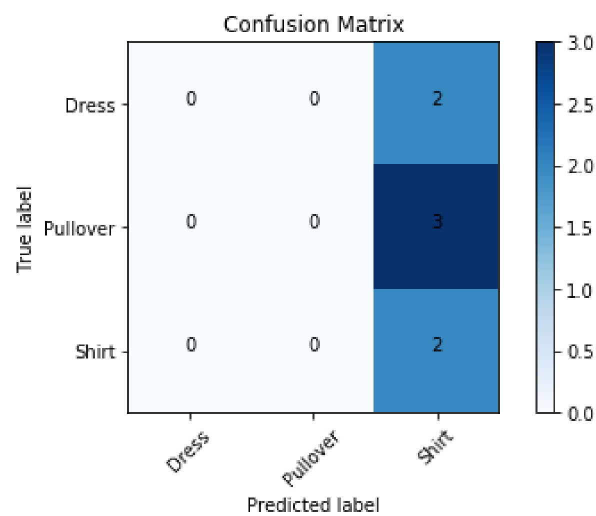

Figure 3 Confusion matrix for basic CNN

The VGG16 model [11] is much more expressive, and by adding the many hidden layers of this convnet which has been pre-trained on 1000 classes of the ImageNet data set, as well as increasing my own dataset from 100 to 1000 images per class, I was able to achieve 78% validation and 76% test accuracy, which is a much better result. VGG16v1 model is likely to achieve accuracy in the range of 72-78% at 95% confidence on an unseen dataset.

Still, there was room to make the model more expressive and bring the results up to 95%.

B. *Final Results*

The three models I evaluated for the final phase of the project are shown in Figure 4, and a graph of the measurement of validation accuracy for all 2x20 training epochs are shown in Figure 5. Once I had adapted the code from Mallick [8], accuracy for VGG16 immediately improved, up to human level. This code included RMSprop for the optimization function, dropout, and a much smaller learning rate. This was extremely exciting.

VGG16 v2 used the 80/20 split of training and validation data and is likely to achieve accuracy in the range of 85-100% at 95% confidence on an unseen dataset.

VGG16 v3 however split the data 50/50 so training data was significantly reduced, and accuracy reduced accordingly. This model is likely to achieve accuracy in the range of 58-91% at 95% confidence on an unseen dataset.

Resnet50 did not perform as well as the VGG16 models. This model is likely to achieve accuracy in the range of 57-80% at 95% confidence on an unseen dataset.

	Basic CNN	VGG16 v1	VGG16 v2	VGG16 v3	ResNet50
Phase	Midterm	Midterm	Final	Final	Final
Train/Dev split	80/20	80/20	80/20	50/50	80/20
Epochs	20	20 x 2	20 x 2	20 x 2	20 x 2
Hidden Layers	1	16	16	16	50
Optimizer function	Adam	Adam	RMSProp	RMSProp	RMSProp
Learning Rate	0.1	.005	2e-4	2e-4	2e-4
Dropout	NA	NA	0.5	0.5	0.5
Activation function	Relu (hidden) SoftMax (final)	Relu (hidden) SoftMax (final)	Relu (hidden) SoftMax (final)	Relu (hidden) SoftMax (final)	Relu (hidden) SoftMax (final)
Loss function	Categorical cross entropy	Categorical cross entropy	Categorical cross entropy	Categorical cross entropy	Categorical cross entropy
Validation test accuracy range with 95% confidence	27-33%	75-78%	85-100%	58-91%	57-80%
Test Accuracy	30%	76%	95%	85%	NA

Figure 4 Final Results

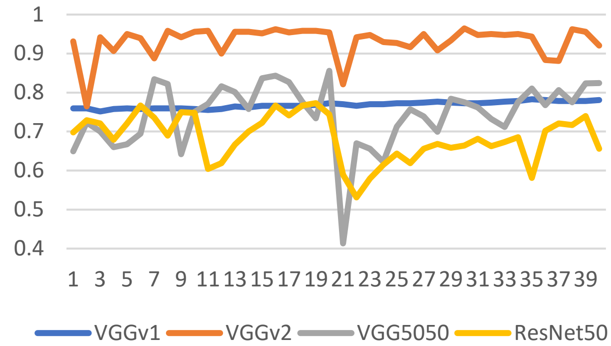

Figure 5 Accuracy over 2 x 20 epochs for each model

VI. DISCUSSION

Basic CNN with limited inputs and only one hidden layer had high bias and essentially only performed with accuracy at the rate of chance.

A deep CNN like VGG16 is much more expressive, and not been overfit as I conducted training on 60% of the data, utilized 20% of the data for a validation set, and tested on 20%. This can be seen by the closeness of accuracy results of validation and test sets and achievement of human level accuracy of 95%. Adding in dropout to the layers drastically improved performance, as well as changing the optimizer from Adam to RMSprop and reducing the learning rate to a much smaller number (see Figure 4). Perhaps further hyperparameter tuning such as learning rate decay might improve the lower bound of the accuracy confidence interval to above 85%, but given the achievement of human level accuracy, I decided to stop here for the purpose of this assignment.

Upon evaluating the errors, it was clear that some classifications are debatable as shown in Figure 5 and 6. Therefore, multiple classes should be assigned to the same image in order for this to work well as a search tool for Google. There was also a repetition of errors through using data augmentation, because when an augmented image was used more than once (with different variations), this multiplied any errors by the same magnitude.

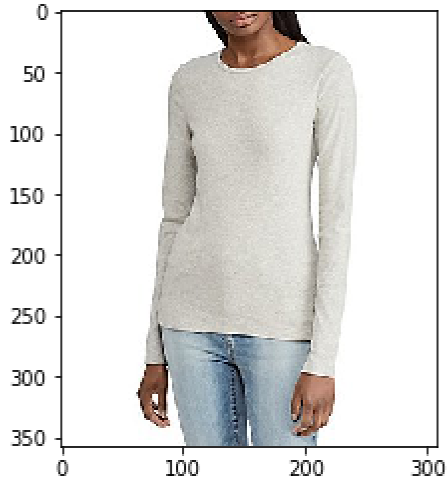

Figure 6 Is this a Shirt or Pullover? Difference in opinion in labelling

Figure 7 Is this a Dress, a Shirt or Pullover? Difference in opinion in labelling

However, the Resnet50 model with even more layers surprisingly did not achieve the same level of performance, so this model implementation may benefit from hyperparameter tuning. Again, for the purpose of this project, I did not continue as VGG16 v2 achieved such great results.

The next phase for this project would be to remove all labels and use my Fashion dataset to explore multiclass Active Learning models [18], and possibly utilize the code developed by Google [19]. This could potentially overcome the high cost of manually labelling images with multiple labels, to account for the differences in opinion in what to label an image. My revised target would be to reduce the variability in the confidence interval, rather than 85-100%, I would like to see a minimum of 95% with 95% confidence.

VII. Conclusion

Based on this analysis of machine learning models focusing on convolutional neural networks, the VGG16 model with dropout (v2) performed the best for classifying fashion images in terms of accuracy and is likely to achieve accuracy in the range of 85-100% at 95% confidence on an unseen dataset. This performance is significantly better than VGG16 v1 without dropout, and Resnet50 for this dataset and therefore the likely performance on future unseen datasets. Further work to develop a multi-class active learning model could improve accuracy even more by increasing the lower bound of the confidence interval to a minimum of 95%.

References

[1] H. Xiao, K. Rasul, and R. Vollgraf, “Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms,” *ArXiv170807747 Cs Stat*, Aug. 2017.

[2] “A VGG-like CNN in keras for Fashion-MNIST with 94% accuracy.” .

[3] “Keras ResNet with image augmentation | Kaggle.” [Online]. Available: https://www.kaggle.com/strali/keras-resnet-with-image-augmentation. [Accessed: 02-Nov-2018].

[4] F. Chollet, “Building powerful image classification models using very little data,” *Keras.io Blog*, 05-Jun-2016. [Online]. Available: https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html. [Accessed: 16-Oct-2018].

[5] H. Vasa, *Python Script to download hundreds of images from “Google Images”. It is a ready-to-run code!: hardikvasa/google-images-download*. 2018.

[6] “Image Preprocessing - Keras Documentation.” [Online]. Available: https://keras.io/preprocessing/image/#imagedatagenerator-methods. [Accessed: 16-Oct-2018].

[7] P. Rodriguez, *Accelerating Deep Learning with Multiprocess Image Augmentation in Keras: stratospark/keras-multiprocess-image-data-generator*. 2018.

[8] S. Mallick, *A toolkit for making real world machine learning and data analysis applications in C++: spmallick/dlib*. 2018.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[10] “ImageNet Tree View.” [Online]. Available: http://image-net.org/explore. [Accessed: 13-Oct-2018].
[11] K. Simonyan and A. Zisserman, “Very Deep CNNS for Large-Scale Visual Recognition,” *arxiv*, 2014. [Online]. Available: https://arxiv-org.libproxy1.usc.edu/pdf/1409.1556.pdf. [Accessed: 16-Oct-2018].
[12] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” *ArXiv151203385 Cs*, Dec. 2015.
[13] T. Dietterich, “Approximate statistical tests for comparing supervised classification learning algorithms,” *Neural Comput.*, vol. 10, no. 7, pp. 1895–1923, 1998.
[14] Y. LeCun, L. Jackel, L. Bottou, A. Brunot, and C. Cortes, “COMPARISON OF LEARNING ALGORITHMS FOR HANDWRITTEN DIGIT RECOGNITION,” p. 9.
[15] deeplizard, *Create and train a CNN Image Classifier with Keras*. .
[16] J. Brownlee, “Gentle Introduction to the Adam Optimization Algorithm for Deep Learning,” *Machine Learning Mastery*, 02-Jul-2017. .
[17] D. Rumsey, “How to Calculate a Confidence Interval for a Population Mean with Unknown Standard Deviation and/or Small Sample Size,” *dummies*. .
[18] Y. Yang, Z. Ma, F. Nie, X. Chang, and A. G. Hauptmann, “Multi-Class Active Learning by Uncertainty Sampling with Diversity Maximization,” *Int. J. Comput. Vis.*, vol. 113, no. 2, pp. 113–127, Jun. 2015.
[19] Google, *Contribute to google/active-learning development by creating an account on GitHub*. Google, 2018.

APPENDICES CODE

A. VGG16v1 and Basic CNN Code

```
# -*- coding: utf-8 -*-
"""
Created on Sat Oct 6 10:51:36 2018

@author: Benjibex
"""

#https://github.com/prashant0598/Keras-Machine-Learning-Deep-Learning-Tutorial

#install required packages
import NumPy as np
import keras
from keras import backend as K
from keras. Models import Sequential
from keras. Layers import Activation
from keras.layers.core import Dense, Flatten
from keras.optimizers import Adam
from keras.metrics import categorical_crossentropy
from keras.preprocessing.image import
ImageDataGenerator
from keras.layers.normalization import BatchNormalization
from keras.layers.convolutional import *
from sklearn.metrics import confusion_matrix
import itertools
import matplotlib.pyplot as plt



train_path =
'C:/Users/Benjibex/Documents/ML_Project/fashion/train'
valid_path =
'C:/Users/Benjibex/Documents/ML_Project/fashion/valid'
test_path =
'C:/Users/Benjibex/Documents/ML_Project/fashion/test'

#just a few images in my prototype so i can test the process
end to end
#target size is 224 here as the tutorial is using vcgconvnet ,
I removed , batch_size=4 from each one because i cannot
see a for loop rolling through
train_batches =
ImageDataGenerator().flow_from_directory(train_path,
target_size=(224,224),
classes=['Dress','Pullover','Shirt'],batch_size=64)
valid_batches =
ImageDataGenerator().flow_from_directory(valid_path,
target_size=(224,224),
classes=['Dress','Pullover','Shirt'],batch_size=16)
test_batches =
ImageDataGenerator().flow_from_directory(test_path,
target_size=(224,224),
classes=['Dress','Pullover','Shirt'],batch_size=20)

# plots images
def plots(ims, figsize=(12,6), rows=1, interp=False,
titles=None):
    if type(ims[0]) is np.ndarray:
        ims = np.array(ims).astype(np.uint8)
        if (ims.shape[-1] != 3):
            ims = ims.transpose((0,2,3,1))
    f = plt.figure(figsize=figsize)
    cols = len(ims)//rows if len(ims) % 2 == 0 else
len(ims)//rows + 1
    for i in range(len(ims)):
        sp = f.add_subplot(rows, cols, i+1)
        sp.axis('Off')
        if titles is not None:
            sp.set_title(titles[i], fontsize=16)
        plt.imshow(ims[i], interpolation=None if interp else
'none')


imgs, labels = next(train_batches)
plots(imgs, titles=labels)



#CNN model changed Dense(n,) where n=number of
classes, in this case 3 classes
model = Sequential([
```

```
    Conv2D(32, (3, 3), activation='relu',
input_shape=(224,224,3)),
    Flatten(),
    Dense(3, activation='softmax'),
  ])

model.compile(Adam(lr=.01),
loss='categorical_crossentropy', metrics=['accuracy'])

model.summary()

model.fit_generator(train_batches, steps_per_epoch=10,
          validation_data=valid_batches,
validation_steps=10, epochs=5, verbose=2)

#predict
test_imgs, test_labels = next(test_batches)
plots(test_imgs, titles=test_labels)

test_labels = test_labels.argmax(axis=1)
test_labels

predictions = model.predict_generator(test_batches,
steps=1, verbose=0)
#predictions = predictions.argmax(axis=1)
for i in predictions:
  print (i)
class_predictions=model.predict_classes(test_batches,
steps=1, verbose=0)

cm = confusion_matrix(test_labels, class_predictions)
classes=['Dress','Pullover','Shirt']
def plot_confusion_matrix(cm, classes,
            normalize=False,
            title='Confusion matrix',
            cmap=plt.cm.Blues):
  """
  This function prints and plots the confusion matrix.
  Normalization can be applied by setting
`normalize=True`.
  """
  plt.imshow(cm, interpolation='nearest', cmap=cmap)
  plt.title(title)
  plt.colorbar()
  tick_marks = np.arange(len(classes))
  plt.xticks(tick_marks, classes, rotation=45)
  plt.yticks(tick_marks, classes)

  if normalize:
    cm = cm.astype('float') / cm.sum(axis=1)[:,
np.newaxis]
    print("Normalized confusion matrix")
  else:
    print('Confusion matrix, without normalization')

  print(cm)


  thresh = cm.max() / 1.
```

```
  for i, j in itertools.product(range(cm.shape[0]),
range(cm.shape[1])):
    plt.text(j, i, cm[i, j],
        horizontalalignment="center",
        color="white" if cm[i, j] > thresh else "black")

  plt.tight_layout()
  plt.ylabel('True label')
  plt.xlabel('Predicted label')
  cm_plot_labels = ['Dress','Pullover','Shirt']
plot_confusion_matrix(cm, cm_plot_labels,
title='Confusion Matrix')

vgg16_model = keras.applications.vgg16.VGG16()
vgg16_model.summary()

type(vgg16_model)

#vgg models use the ReLu activation function
model = Sequential()
for layer in vgg16_model.layers:
  model.add(layer)

model.summary()
#remove the last layer with the 1000 classes so you can
adapt it to the number of classes you need
model.layers.pop()
model.summary()

#freeze the weights in the vgg model as we dont want it to
change from the weights from the original 1000 classes
for layer in model.layers:
  layer.trainable = False

# numerical needs to be the number of categories in this
case 3
model.add(Dense(3, activation='softmax'))
model.summary()

model.compile(Adam(lr=.005),
loss='categorical_crossentropy', metrics=['accuracy'])

model.fit_generator(train_batches, steps_per_epoch=10,
          validation_data=valid_batches,
validation_steps=10, epochs=20, verbose=2)


test_imgs, test_labels = next(test_batches)
plots(test_imgs, titles=test_labels)


test_labels
y_classes_true = test_labels.argmax(axis=-1)
y_classes_true

predictions = model.predict_generator(test_batches,
steps=1, verbose=0)

y_classes_pred = predictions.argmax(axis=-1)
y_classes_pred
```

```
accuracy =
keras.metrics.categorical_accuracy(y_classes_true,
y_classes_pred)
print(accuracy)
accuracy

cm = confusion_matrix(y_classes_true, y_classes_pred])

classes=['Dress','Pullover','Shirt']

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting
`normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:,
np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)



    thresh = cm.max() / 1.
    for i, j in itertools.product(range(cm.shape[0]),
range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    cm_plot_labels = ['Dress','Pullover','Shirt']
plot_confusion_matrix(cm, cm_plot_labels,
title='Confusion Matrix')

labels
```

B. VGG16v 2 code

```
# -*- coding: utf-8 -*-
"""
Created on Mon Nov 12 11:45:37 2018

@author: Benjibex
"""

#based on the tutorial by Satya Mallick
https://github.com/spmallick/learnopencv/blob/master/Kera
s-Transfer-Learning/transfer-learning-vgg.ipynb
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from __future__ import print_function
import keras
from keras.utils import to_categorical
import os
from keras.preprocessing.image import
ImageDataGenerator, load_img
from sklearn.metrics import confusion_matrix
from keras.applications import VGG16

vgg_conv = VGG16(weights='imagenet',
                 include_top=False,
                 input_shape=(224, 224, 3))

vgg_conv.summary()


train_dir = './Documents/ML_Project/fashion/train'
validation_dir = './Documents/ML_Project/fashion/valid'
test_dir = './Documents/ML_Project/fashion/test'

nTrain = 1920
nVal = 480
nTest = 600

#adding data augmentation, before it was the imagerescale
only
datagen = ImageDataGenerator(#        zoom_range=0.2, #
randomly zoom into images
#        rotation_range=10,  # randomly rotate images in the
range (degrees, 0 to 180)
#        width_shift_range=0.1,  # randomly shift images
horizontally (fraction of total width)
#        height_shift_range=0.1,  # randomly shift images
vertically (fraction of total height)
#        horizontal_flip=True,  # randomly flip images
#        vertical_flip=False, # randomly flip
        rescale=1./255) #rescale images to be between 0 and 1
batch_size = 20

train_features = np.zeros(shape=(nTrain, 7, 7, 512))
train_labels = np.zeros(shape=(nTrain,3))

train_generator = datagen.flow_from_directory(
    train_dir,
    target_size=(224, 224),
    batch_size=batch_size,
    class_mode='categorical',
    shuffle='shuffle')


i = 0
```

```
for inputs_batch, labels_batch in train_generator:
    features_batch = vgg_conv.predict(inputs_batch)
    train_features[i * batch_size : (i + 1) * batch_size] =
features_batch
    train_labels[i * batch_size : (i + 1) * batch_size] =
labels_batch
    i += 1
    if i * batch_size >= nTrain:
        break

train_features = np.reshape(train_features, (nTrain, 7 * 7 *
512))

validation_features = np.zeros(shape=(nVal, 7, 7, 512))
validation_labels = np.zeros(shape=(nVal,3))

validation_generator = datagen.flow_from_directory(
    validation_dir,
    target_size=(224, 224),
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=False)

i = 0
for inputs_batch, labels_batch in validation_generator:
    features_batch = vgg_conv.predict(inputs_batch)
    validation_features[i * batch_size : (i + 1) * batch_size]
= features_batch
    validation_labels[i * batch_size : (i + 1) * batch_size] =
labels_batch
    i += 1
    if i * batch_size >= nVal:
        break

validation_features = np.reshape(validation_features, (nVal,
7 * 7 * 512))


from keras import models
from keras import layers
from keras import optimizers

model = models.Sequential()
model.add(layers.Dense(256, activation='relu', input_dim=7
* 7 * 512))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(3, activation='softmax'))

model.compile(optimizer=optimizers.RMSprop(lr=2e-4),
              loss='categorical_crossentropy',
              metrics=['acc'])

history = model.fit(train_features,
                    train_labels,
                    epochs=20,
                    batch_size=batch_size,

validation_data=(validation_features,validation_labels))

# Plot the accuracy and loss curves
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'b', label='Training acc')
plt.plot(epochs, val_acc, 'r', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()

#examine the errors
fnames = validation_generator.filenames

ground_truth = validation_generator.classes

label2index = validation_generator.class_indices

# Getting the mapping from class index to class label
idx2label = dict((v,k) for k,v in label2index.items())

predictions = model.predict_classes(validation_features)
prob = model.predict(validation_features)

errors = np.where(predictions != ground_truth)[0]
print("No of errors = {}/{}".format(len(errors),nVal))

for i in range(len(errors)):
    pred_class = np.argmax(prob[errors[i]])
    pred_label = idx2label[pred_class]

    print('Original label:{}, Prediction :{}, confidence :
{:.3f}'.format(
        fnames[errors[i]].split('/')[0],
        pred_label,
        prob[errors[i]][pred_class]))

    original =
load_img('{}/{}'.format(validation_dir,fnames[errors[i]]))
    plt.imshow(original)
    plt.show()


#flow the test images through now so we can make
predictions on the test data
test_features = np.zeros(shape=(nTest, 7, 7, 512))
test_labels = np.zeros(shape=(nTest,3))

test_generator = datagen.flow_from_directory(
    test_dir,
```

```
    target_size=(224, 224),
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=False)

i = 0
for inputs_batch, labels_batch in test_generator:
    features_batch = vgg_conv.predict(inputs_batch)
    test_features[i * batch_size : (i + 1) * batch_size] =
features_batch
    test_labels[i * batch_size : (i + 1) * batch_size] =
labels_batch
    i += 1
    if i * batch_size >= nTest:
        break

test_features = np.reshape(test_features, (nTest, 7 * 7 *
512))

#make predictions on the test data

ground_truth_test = test_generator.classes

predictions_test = model.predict_classes(test_features)
prob_test = model.predict(test_features)


errors_test = np.where(predictions_test !=
ground_truth_test)[0]
print("No of test errors =
{}/{}".format(len(errors_test),nTest))
```

RESNET50 CODE

```
# -*- coding: utf-8 -*-
"""
Created on Mon Nov 12 11:45:37 2018

@author: Benjibex
"""

#based on the tutorial by Satya Mallick
https://github.com/spmallick/learnopencv/blob/master/Kera
s-Transfer-Learning/transfer-learning-vgg.ipynb
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from __future__ import print_function
import keras
from keras.utils import to_categorical
import os
from keras.preprocessing.image import
ImageDataGenerator, load_img

from keras.applications import resnet50

res_conv = resnet50.ResNet50(weights='imagenet',
              include_top=False,
              input_shape=(224, 224, 3))

res_conv.summary()

train_dir = './Documents/ML_Project/fashion/train'
validation_dir = './Documents/ML_Project/fashion/valid'
test_dir = './Documents/ML_Project/fashion/test'

nTrain = 1920
nVal = 480
nTest = 600

#adding data augmentation, before it was the imagerescale
only
datagen = ImageDataGenerator(#        zoom_range=0.2, #
randomly zoom into images
#        rotation_range=10,  # randomly rotate images in the
range (degrees, 0 to 180)
        width_shift_range=0.1,  # randomly shift images
horizontally (fraction of total width)
        height_shift_range=0.1,  # randomly shift images
vertically (fraction of total height)
        horizontal_flip=True,  # randomly flip images
        vertical_flip=False, # randomly flip
        rescale=1./255) #rescale images to be between 0 and 1
batch_size = 20

train_features = np.zeros(shape=(nTrain, 7, 7, 2048))
train_labels = np.zeros(shape=(nTrain,3))

train_generator = datagen.flow_from_directory(
    train_dir,
    target_size=(224, 224),
    batch_size=batch_size,
    class_mode='categorical',
    shuffle='shuffle')


i = 0
for inputs_batch, labels_batch in train_generator:
    features_batch = res_conv.predict(inputs_batch)
    train_features[i * batch_size : (i + 1) * batch_size] =
features_batch
    train_labels[i * batch_size : (i + 1) * batch_size] =
labels_batch
    i += 1
    if i * batch_size >= nTrain:
        break

train_features = np.reshape(train_features, (nTrain, 7 * 7 *
2048))

validation_features = np.zeros(shape=(nVal, 7, 7, 2048))
validation_labels = np.zeros(shape=(nVal,3))

validation_generator = datagen.flow_from_directory(
    validation_dir,
    target_size=(224, 224),
    batch_size=batch_size,
    class_mode='categorical',
```

```
    shuffle=False)

i = 0
for inputs_batch, labels_batch in validation_generator:
    features_batch = res_conv.predict(inputs_batch)
    validation_features[i * batch_size : (i + 1) * batch_size]
= features_batch
    validation_labels[i * batch_size : (i + 1) * batch_size] =
labels_batch
    i += 1
    if i * batch_size >= nVal:
        break

validation_features = np.reshape(validation_features, (nVal,
7 * 7 * 2048))



from keras import models
from keras import layers
from keras import optimizers

model = models.Sequential()
model.add(layers.Dense(256, activation='relu', input_dim=7
* 7 * 2048))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(3, activation='softmax'))

model.compile(optimizer=optimizers.RMSprop(lr=2e-4),
              loss='categorical_crossentropy',
              metrics=['acc'])

history = model.fit(train_features,
                    train_labels,
                    epochs=20,
                    batch_size=batch_size,

validation_data=(validation_features,validation_labels))

# Plot the accuracy and loss curves
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'b', label='Training acc')
plt.plot(epochs, val_acc, 'r', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()

#examine the errors
fnames = validation_generator.filenames

ground_truth = validation_generator.classes

label2index = validation_generator.class_indices

# Getting the mapping from class index to class label
idx2label = dict((v,k) for k,v in label2index.items())

predictions = model.predict_classes(validation_features)
prob = model.predict(validation_features)

errors = np.where(predictions != ground_truth)[0]
print("No of errors = {}/{}".format(len(errors),nVal))

for i in range(len(errors)):
    pred_class = np.argmax(prob[errors[i]])
    pred_label = idx2label[pred_class]

    print('Original label:{}, Prediction :{}, confidence :
{:.3f}'.format(
        fnames[errors[i]].split('/')[0],
        pred_label,
        prob[errors[i]][pred_class]))

    original =
load_img('{}/{}'.format(validation_dir,fnames[errors[i]]))
    plt.imshow(original)
    plt.show()

#make predictions i cant get this code to work here for
some reason
#flow the test images through now so we can make
predictions on the test data
test_features = np.zeros(shape=(nTest, 7, 7, 2048))
test_labels = np.zeros(shape=(nTest,3))

test_generator = datagen.flow_from_directory(
    test_dir,
    target_size=(224, 224),
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=False)

i = 0
for inputs_batch, labels_batch in test_generator:
    features_batch = vgg_conv.predict(inputs_batch)
    test_features[i * batch_size : (i + 1) * batch_size] =
features_batch
    test_labels[i * batch_size : (i + 1) * batch_size] =
labels_batch
    i += 1
    if i * batch_size >= nTest:
        break

test_features = np.reshape(test_features, (nTest, 7 * 7 *
2048))
```

```
#make predictions on the test data

ground_truth_test = test_generator.classes

predictions_test = model.predict_classes(test_features)
prob_test = model.predict(test_features)


errors_test = np.where(predictions_test !=
ground_truth_test)[0]
print("No of test errors =
{}/{}".format(len(errors_test),nTest))
```