# Train a simple convnet on the Fashion MNIST dataset

In this, we will see how to deal with image data and train a convnet for image classification task.

## Load the `fashion_mnist` dataset

** Use keras.datasets to load the dataset **

```
from keras.datasets import fashion_mnist
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
```

## Find no.of samples are there in training and test datasets

```
import tensorflow.compat.v1 as tf1
tf1.disable_v2_behavior()
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python/co
Instructions for updating:
non-resource variables are not supported in the long term
```

```
from __future__ import absolute_import, division, print_function
import numpy as np
import keras
from keras.datasets import cifar10, mnist
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout, Flatten, Reshape
from keras.layers import Convolution2D, MaxPooling2D
from keras.utils import np_utils
import pickle
from matplotlib import pyplot as plt
import seaborn as sns
plt.rcParams['figure.figsize'] = (15, 8)
```

```
from google.colab import drive
drive.mount('/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=9473189

Enter your authorization code:
··········
Mounted at /drive
```

```
%matplotlib inline
# Load/Prep the Data
(x_train, y_train_num), (x_test, y_test_num) = mnist.load_data()
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1).astype('float32')
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1).astype('float32')
x_train /= 255
x_test /= 255
y_train = np_utils.to_categorical(y_train_num, 10)
y_test = np_utils.to_categorical(y_test_num, 10)

print('--- THE DATA ---')
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
```

```
Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz
11493376/11490434 [==============================] - 1s 0us/step
--- THE DATA ---
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
```

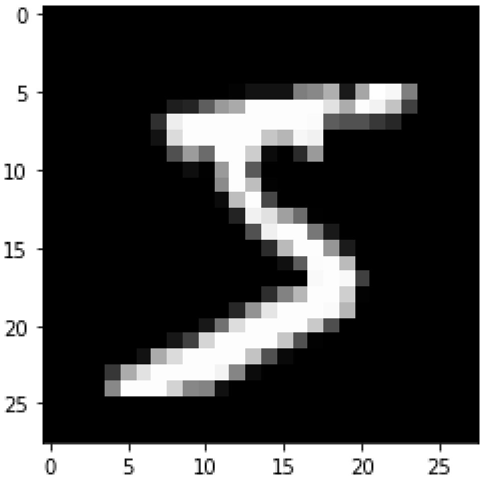
```
""
```

## Find dimensions of an image in the dataset

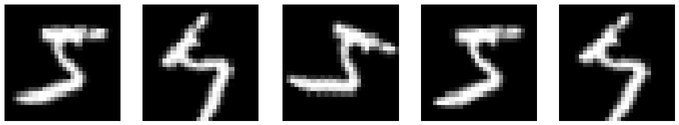
```
from keras.preprocessing.image import ImageDataGenerator

# This will do preprocessing and realtime data augmentation:
datagen = ImageDataGenerator(
    featurewise_center=False,  # set input mean to 0 over the dataset
    samplewise_center=False,  # set each sample mean to 0
    featurewise_std_normalization=False,  # divide inputs by std of the dataset
    samplewise_std_normalization=False,  # divide each input by its std
    zca_whitening=False,  # apply ZCA whitening
    rotation_range=50,  # randomly rotate images in the range (degrees, 0 to 180)
    width_shift_range=0.01,  # randomly shift images horizontally (fraction of total width)
    height_shift_range=0.01,  # randomly shift images vertically (fraction of total height)
    horizontal_flip=False,  # randomly flip images
    vertical_flip=False)  # randomly flip images

# Prepare the generator
datagen.fit(x_train)
```

```
plt.imshow(x_train[0].squeeze(), cmap='gray')
plt.plot()
```

```
[]
```



```
gen = datagen.flow(x_train[:1], batch_size=1)
for i in range(1, 6):
    plt.subplot(1,5,i)
    plt.axis("off")
    plt.imshow(gen.next().squeeze(), cmap='gray')
    plt.plot()
```



## Convert train and test labels to one hot vectors

** check `keras.utils.to_categorical()` **

```
from numpy import array
from numpy import argmax
from keras.utils import to_categorical
# define example
train_data = [60000,28,28,1]
train_data = array(train_data)
print(train_data)
# one hot encode
encoded = to_categorical(train_data)
print(encoded)
# invert encoding
inverted = argmax(encoded[0])
print(inverted)
```

```
[60000    28    28     1]
[[0. 0. 0. ... 0. 0. 1.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 1. 0. ... 0. 0. 0.]]
60000
```

```
from numpy import array
from numpy import argmax
from keras.utils import to_categorical
# define example
test_data = [10000]
test_data = array(test_data)
print(test_data)
# one hot encode
encoded = to_categorical(test_data)
print(encoded)
# invert encoding
inverted = argmax(encoded[0])
print(inverted)
```

```
[10000]
[[0. 0. 0. ... 0. 0. 1.]]
10000
```

## Normalize both the train and test image data from 0-255 to 0-1

```
# example of using ImageDataGenerator to normalize images
from keras.datasets import mnist
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.preprocessing.image import ImageDataGenerator
# load dataset
(trainX, trainY), (testX, testY) = mnist.load_data()
# reshape dataset to have a single channel
width, height, channels = trainX.shape[1], trainX.shape[2], 1
trainX = trainX.reshape((trainX.shape[0], width, height, channels))
testX = testX.reshape((testX.shape[0], width, height, channels))
# one hot encode target values
trainY = to_categorical(trainY)
testY = to_categorical(testY)
# confirm scale of pixels
print('Train min=%.3f, max=%.3f' % (trainX.min(), trainX.max()))
print('Test min=%.3f, max=%.3f' % (testX.min(), testX.max()))
# create generator (1.0/255.0 = 0.003921568627451)
```

```
datagen = ImageDataGenerator(rescale=1.0/255.0)
# prepare an iterators to scale images
train_iterator = datagen.flow(trainX, trainY, batch_size=64)
test_iterator = datagen.flow(testX, testY, batch_size=64)
print('Batches train=%d, test=%d' % (len(train_iterator), len(test_iterator)))
# confirm the scaling works
batchX, batchy = train_iterator.next()
print('Batch shape=%s, min=%.3f, max=%.3f' % (batchX.shape, batchX.min(), batchX.max()))
# define model
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(width, height, channels)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(10, activation='softmax'))
# compile model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
# fit model with generator
model.fit_generator(train_iterator, steps_per_epoch=len(train_iterator), epochs=5)
# evaluate model
_, acc = model.evaluate_generator(test_iterator, steps=len(test_iterator), verbose=0)
print('Test Accuracy: %.3f' % (acc * 100))
```

```
Train min=0.000, max=255.000
Test min=0.000, max=255.000
Batches train=938, test=157
Batch shape=(64, 28, 28, 1), min=0.000, max=1.000
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/optimizers.py:793:

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python/op
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_

Epoch 1/5
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_

938/938 [==============================] - 16s 18ms/step - loss: 0.1711 - acc: 0.9483
Epoch 2/5
938/938 [==============================] - 9s 10ms/step - loss: 0.0529 - acc: 0.9840
Epoch 3/5
938/938 [==============================] - 9s 10ms/step - loss: 0.0377 - acc: 0.9883
Epoch 4/5
938/938 [==============================] - 9s 10ms/step - loss: 0.0285 - acc: 0.9911
Epoch 5/5
938/938 [==============================] - 9s 10ms/step - loss: 0.0219 - acc: 0.9930
Test Accuracy: 99.040
```

```
train_data.shape
```

```
(4,)
```

```
train_data = keras.utils.np_utils.to_categorical(train_data)
test_data = keras.utils.np_utils.to_categorical(train_data)
```

## Reshape the data from 28x28 to 28x28x1 to match input

dimensions in Conv2D layer in keras

```
# done in the above step when reshaped to 255
```

## Import the necessary layers from keras to build the model

```
y_train = keras.utils.np_utils.to_categorical(y_train)
y_test = keras.utils.np_utils.to_categorical(y_train)
```

Build a model

** with 2 Conv layers having `32 3x3 filters` in both convolutions with `relu activations` and `flatt` map into 2 fully connected layers (or Dense Layers) having 128 and 10 neurons with `relu` and `softma` using `categorical_crossentropy` loss with `adam` optimizer train the model with early stopping `patie`

```
TRAIN = False
BATCH_SIZE = 32
EPOCHS = 10
```

```
model1 = Sequential()

# 1st Conv Layer
model1.add(Convolution2D(32, 3, 3, input_shape=(28, 28, 1)))
model1.add(Activation('relu'))

# 2nd Conv Layer
model1.add(Convolution2D(32, 3, 3))
model1.add(Activation('relu'))

# Fully Connected Layer
model1.add(Flatten())
model1.add(Dense(128))
model1.add(Activation('relu'))

# Prediction Layer
model1.add(Dense(10))
model1.add(Activation('softmax'))

# Loss and Optimizer
model1.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
model1.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Store Training Results
early_stopping = keras.callbacks.EarlyStopping(monitor='val_acc', patience=5, verbose=1, mode
callback_list = [early_stopping]

# Train the model2
model1.fit(x_train, y_train, batch_size=BATCH_SIZE, nb_epoch=EPOCHS,
              validation_data=(x_test, y_test), callbacks=callback_list)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:4: UserWarning: Update your
  after removing the cwd from sys.path.
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:8: UserWarning: Update your

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:29: UserWarning: The `nb_ep
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-104-8b9e997cf617> in <module>()
     27 # Train the model2
     28 model1.fit(x_train, y_train, batch_size=BATCH_SIZE, nb_epoch=EPOCHS,
---> 29               validation_data=(x_test, y_test), callbacks=callback_list)

                              2 frames
/usr/local/lib/python3.6/dist-packages/keras/engine/training_utils.py in standardize_inp
    129                             ': expected ' + names[i] + ' to have ' +
    130                             str(len(shape)) + ' dimensions, but got array '
--> 131                             'with shape ' + str(data_shape))
    132                 if not check_batch_axis:
    133                     data_shape = data_shape[1:]

ValueError: Error when checking target: expected activation_31 to have 2 dimensions, but
```

SEARCH STACK OVERFLOW

## Now, to the above model add `max` pooling layer of `filter size 2x2` and `dropout` the 2 conv layers and run the model

```
BATCH_SIZE = 32
EPOCHS = 10



# Define Model
model2 = Sequential()

# 1st Conv Layer
model2.add(Convolution2D(32, 3, 3, input_shape=(28, 28, 1)))
model2.add(Activation('relu'))

# 2nd Conv Layer
model2.add(Convolution2D(32, 3, 3))
```

```
model2.add(Activation('relu'))

# Max Pooling
model2.add(MaxPooling2D(pool_size=(2,2)))

# Dropout
model2.add(Dropout(0.25))

# Fully Connected Layer
model2.add(Flatten())
model2.add(Dense(128))
model2.add(Activation('relu'))

# Prediction Layer
model2.add(Dense(10))
model2.add(Activation('softmax'))

# Loss and Optimizer
model2.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Store Training Results
early_stopping = keras.callbacks.EarlyStopping(monitor='val_acc', patience=5, verbose=1, mode
callback_list = [early_stopping]

# Train the model
model2.fit(x_train, y_train, batch_size=BATCH_SIZE, nb_epoch=EPOCHS,
              validation_data=(x_test, y_test), callbacks=callback_list)
     WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_
     Instructions for updating:
     Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
     /usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:4: UserWarning: Update your
       after removing the cwd from sys.path.
     /usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:8: UserWarning: Update your

     /usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:35: UserWarning: The `nb_ep
     ---------------------------------------------------------------------------
     ValueError                                Traceback (most recent call last)
     <ipython-input-86-44879a4f5261> in <module>()
          33 # Train the model
          34 model2.fit(x_train, y_train, batch_size=BATCH_SIZE, nb_epoch=EPOCHS,
     ---> 35                validation_data=(x_test, y_test), callbacks=callback_list)

                                          2 frames

     /usr/local/lib/python3.6/dist-packages/keras/engine/training_utils.py in standardize_inp
         129                             ': expected ' + names[i] + ' to have ' +
         130                             str(len(shape)) + ' dimensions, but got array '
     --> 131                             'with shape ' + str(data_shape))
         132                 if not check_batch_axis:
         133                     data_shape = data_shape[1:]

     ValueError: Error when checking target: expected activation_23 to have 2 dimensions, but
```

SEARCH STACK OVERFLOW

## Now, to the above model, lets add Data Augmentation

## Import the ImageDataGenrator from keras and fit the training images

```
datagen = ImageDataGenerator(
    featurewise_center=False,  # set input mean to 0 over the dataset
    samplewise_center=False,  # set each sample mean to 0
    featurewise_std_normalization=False,  # divide inputs by std of the dataset
    samplewise_std_normalization=False,  # divide each input by its std
    zca_whitening=False,  # apply ZCA whitening
    rotation_range=50,  # randomly rotate images in the range (degrees, 0 to 180)
    width_shift_range=0.5,  # randomly shift images horizontally (fraction of total width)
    height_shift_range=0.5,  # randomly shift images vertically (fraction of total height)
    horizontal_flip=False,  # randomly flip images
    vertical_flip=False)  # randomly flip images

# Prepare the generator
datagen.fit(x_train)
```

## Showing 5 versions of the first image in training dataset using image datagenerator.flow()

```
from matplotlib import pyplot as plt
gen = datagen.flow(x_train[0:1], batch_size=1)
for i in range(1, 6):
    plt.subplot(1,5,i)
    plt.axis("off")
    plt.imshow(gen.next().squeeze(), cmap='gray')
    plt.plot()
plt.show()
```

## Run the above model using fit_generator()

```
model2.fit_generator(datagen.flow(x_train, y_train,batch_size=32),
                    samples_per_epoch=x_train.shape[0],
                    nb_epoch=10,
                    validation_data=(x_test, y_test), callbacks=callback_list)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:4: UserWarning: The semanti
  after removing the cwd from sys.path.
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:4: UserWarning: Update your
  after removing the cwd from sys.path.
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-106-913d9056826b> in <module>()
      2                     samples_per_epoch=x_train.shape[0],
      3                     nb_epoch=10,
----> 4                     validation_data=(x_test, y_test), callbacks=callback_list)

                              4 frames
/usr/local/lib/python3.6/dist-packages/keras/engine/training_utils.py in standardize_inp
    129                             ': expected ' + names[i] + ' to have ' +
    130                             str(len(shape)) + ' dimensions, but got array '
--> 131                             'with shape ' + str(data_shape))
    132                 if not check_batch_axis:
    133                     data_shape = data_shape[1:]

ValueError: Error when checking target: expected activation_23 to have 2 dimensions, but
```

SEARCH STACK OVERFLOW

## Report the final train and validation accuracy

```
loss_and_metrics = model2.evaluate(x_test, y_test)
print(loss_and_metrics)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-105-8701d52fafb6> in <module>()
----> 1 loss_and_metrics = model2.evaluate(x_test, y_test)
      2 print(loss_and_metrics)
```

2 frames

```
/usr/local/lib/python3.6/dist-packages/keras/engine/training_utils.py in standardize_inp
    129                             ': expected ' + names[i] + ' to have ' +
    130                             str(len(shape)) + ' dimensions, but got array '
--> 131                             'with shape ' + str(data_shape))
    132                 if not check_batch_axis:
    133                     data_shape = data_shape[1:]

ValueError: Error when checking target: expected activation_23 to have 2 dimensions, but
```

SEARCH STACK OVERFLOW

## DATA AUGMENTATION ON CIFAR10 DATASET

One of the best ways to improve the performance of a Deep Learning model is to add more data to th
gathering more instances from the wild that are representative of the distinction task, we want to dev
enhance the data we already have. There are many ways to augment existing datasets and produce m
domain, these are done to utilize the full power of the convolutional neural network, which is able to c
This translational invariance is what makes image recognition such a difficult task in the first place. Y
representative of the many different positions, angles, lightings, and miscellaneous distortions that ar

### Import neessary libraries for data augmentation

```
# Already imported above
```

### Load CIFAR10 dataset

```
from keras.datasets import cifar10
(x_train_cifar, y_train_cifar), (x_test_cifar, y_test_cifar) = cifar10.load_data()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 6s 0us/step
```

```
x_train_cifar.shape
```

```
(50000, 32, 32, 3)
```

```
x_test_cifar.shape
```

```
(10000, 32, 32, 3)
```

```
y_train_cifar.shape
```

```
(50000, 1)
```

```
y_test_cifar.shape
```

```
(10000, 1)
```

```
y_train_cifar[0].shape
```

```
(1,)
```

```
y_train[0][0]
```

```
array([1., 0.], dtype=float32)
```

```
x_train_cifar = x_train_cifar.astype('float32')
x_test_cifar = x_test_cifar.astype('float32')
x_train_cifar /= 255
x_test_cifar /= 255
```

## Create a data_gen funtion to genererator with image rotation,shifting image hori with random flip horizontally.

```
datagen = ImageDataGenerator(
    featurewise_center=False,  # set input mean to 0 over the dataset
    featurewise_center=False,  # set input mean to 0 over the dataset
    samplewise_center=False,  # set each sample mean to 0
    featurewise_std_normalization=False,  # divide inputs by std of the dataset
    samplewise_std_normalization=False,  # divide each input by its std
    zca_whitening=False,  # apply ZCA whitening
    rotation_range=50,  # randomly rotate images in the range (degrees, 0 to 180)
    width_shift_range=0.5,  # randomly shift images horizontally (fraction of total width)
    height_shift_range=0.5,  # randomly shift images vertically (fraction of total height)
    horizontal_flip=True,  # randomly flip images
    vertical_flip=False)  # randomly flip images
```

## Prepare/fit the generator.

```
datagen.fit(x_train_cifar)
```

## Generate 5 images for 1 of the image of CIFAR10 train dataset.

```
from matplotlib import pyplot as plt
gen = datagen.flow(x_train_cifar[0:1], batch_size=1)
for i in range(1, 6):
    plt.subplot(1,5,i)
    plt.axis("off")
    plt.imshow(gen.next().squeeze(), cmap='gray')
    plt.plot()
plt.show()
```