```
from sklearn import datasets
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout
from keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from keras.wrappers.scikit_learn import KerasClassifier, KerasRegressor
from sklearn.model_selection import GridSearchCV, StratifiedShuffleSplit
from sklearn.preprocessing import StandardScaler
import pandas as pd
import numpy as np
```

```
Using TensorFlow backend.
```

```
# Load the iris dataset

iris = datasets.load_iris()
```

```
# Store predictive features and the X and y variable

X = iris.data
y = iris.target
```

```
# Split the dataset into the training set and test set and take necessary transfor

X_train, X_test, y_train, y_test = train_test_split(X,y, stratify=y)
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
ss = StandardScaler()
X_test_ss = ss.fit_transform(X_test)
X_train_ss = ss.fit_transform(X_train)
```

```
# Create a function called make_model for GridSearch

def make_model(optimizer="adam", hidden_size=5,drop_out=0.0):
    model = Sequential([
      Dense(hidden_size, input_shape=(4,)),
      Activation('relu'),
      Dense(8),
      Dropout(drop_out),
      Activation('relu'),
      Dense(3),
      Dropout(drop_out),
      Activation("softmax")
    ])
    model.compile(optimizer=optimizer,loss="categorical_crossentropy",
                  metrics=['accuracy'])
    return model
clf = KerasClassifier(make_model)
```

```
# Fit the classifier to the data

clf.fit(X_train_ss, y_train, epochs=10)
```

```
# GridSearch for the best tuning parameters

param_grid = {'epochs': np.arange(50,80,10),
              'hidden_size': [64,128,256],
              'drop_out' : [0.1, 0.25, 0.5]
             }


cv = StratifiedShuffleSplit(n_splits=2, test_size=0.5, random_state=42)
grid = GridSearchCV(clf, param_grid=param_grid, cv=cv)
```

```
grid.fit(X_train_ss, y_train)
```

```
112/112 [==============================] - 0s 207us/step - loss: 0.3158 - ac
Epoch 29/70
112/112 [==============================] - 0s 184us/step - loss: 0.2822 - ac
Epoch 30/70
112/112 [==============================] - 0s 197us/step - loss: 0.3067 - ac
Epoch 31/70
112/112 [==============================] - 0s 174us/step - loss: 0.2746 - ac
Epoch 32/70
112/112 [==============================] - 0s 243us/step - loss: 0.2791 - ac
Epoch 33/70
112/112 [==============================] - 0s 180us/step - loss: 0.2713 - ac
Epoch 34/70
112/112 [==============================] - 0s 159us/step - loss: 0.2256 - ac
Epoch 35/70
112/112 [==============================] - 0s 175us/step - loss: 0.2882 - ac
Epoch 36/70
112/112 [==============================] - 0s 202us/step - loss: 0.2388 - ac
Epoch 37/70
112/112 [==============================] - 0s 204us/step - loss: 0.2231 - ac
Epoch 38/70
112/112 [==============================] - 0s 161us/step - loss: 0.2084 - ac
Epoch 39/70
112/112 [==============================] - 0s 175us/step - loss: 0.2372 - ac
Epoch 40/70
112/112 [==============================] - 0s 175us/step - loss: 0.2235 - ac
Epoch 41/70
112/112 [==============================] - 0s 186us/step - loss: 0.1848 - ac
Epoch 42/70
112/112 [==============================] - 0s 179us/step - loss: 0.2222 - ac
Epoch 43/70
112/112 [==============================] - 0s 190us/step - loss: 0.1907 - ac
Epoch 44/70
112/112 [==============================] - 0s 182us/step - loss: 0.2031 - ac
Epoch 45/70
112/112 [==============================] - 0s 212us/step - loss: 0.1948 - ac
Epoch 46/70
```

Double-click (or enter) to edit

```
112/112 [==============================] - 0s 188us/step - loss: 0.2334 - ac
```

```
# Create a dataframe to show model performances depended on tuning parameters

res = pd.DataFrame(grid.cv_results_)
res.pivot_table(index=["param_epochs", "param_hidden_size","param_drop_out" ],
                values=['mean_train_score', "mean_test_score"])
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: Fut
  warnings.warn(*warn_args, **warn_kwargs)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: Fut
  warnings.warn(*warn_args, **warn_kwargs)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: Fut
  warnings.warn(*warn_args, **warn_kwargs)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: Fut
  warnings.warn(*warn_args, **warn_kwargs)
```

			mean_test_score	mean_train
param_epochs	param_hidden_size	param_drop_out		
50	**64**	**0.10**	0.785714	0
		0.25	0.883929	0
		0.50	0.857143	0
	128	**0.10**	0.901786	0
		0.25	0.821429	0
		0.50	0.892857	0
	256	**0.10**	0.937500	0
		0.25	0.857143	0
		0.50	0.910714	0
60	**64**	**0.10**	0.785714	0
		0.25	0.883929	0
		0.50	0.830357	0
	128	**0.10**	0.928571	0
		0.25	0.946429	0
		0.50	0.901786	0
	256	**0.10**	0.910714	0
		0.25	0.892857	0
		0.50	0.910714	0
70	**64**	**0.10**	0.892857	0

```
# Extract the best tuning parameter combination
grid.best_params_
```

```
{'drop_out': 0.1, 'epochs': 70, 'hidden_size': 256}
```

```
# Store the best test scoret
score = grid.score(X_test_ss, y_test)
```

```
38/38 [==============================] - 2s 54ms/step
```

```
# Print the best test score
```

```
score
```

```
0.9736842105263158
```

```
# Import necessary libraries

from sklearn import datasets
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout
from keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from keras.wrappers.scikit_learn import KerasClassifier, KerasRegressor
from sklearn.model_selection import GridSearchCV, StratifiedShuffleSplit
from sklearn.preprocessing import StandardScaler
import pandas as pd
import numpy as np
from keras.datasets import fashion_mnist
from keras.utils.np_utils import to_categorical
from keras.wrappers.scikit_learn import KerasClassifier, KerasRegressor
from sklearn.model_selection import GridSearchCV
```

```
import tensorflow as tf
device_name = tf.test.gpu_device_name()
if device_name != '/device:GPU:0':
  raise SystemError('GPU device not found')
print('Found GPU at: {}'.format(device_name))
```

```
Found GPU at: /device:GPU:0
```

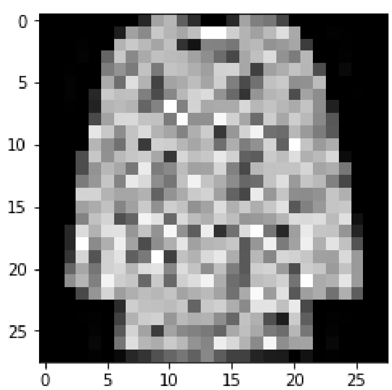
```
# Load the dataset
# 60000 samples, 28 x 28 pixel

((X_train, y_train), (X_test, y_test)) = fashion_mnist.load_data()
```

```
# Set a random seed for subsampling

idx = np.random.randint(1,60000,size=10000)
```

```
# Subsample 10,000 samples out of 60,000 samples

X_sub_train = X_train[idx]
y_sub_train = y_train[idx]
```

```
# Check if a corresponding matrix is aligned correctly by visualization

import matplotlib.pyplot as plt
import matplotlib.image as mpimg

plt.imshow(X_sub_train[0], cmap=plt.get_cmap('gray'))
```

```
<matplotlib.image.AxesImage at 0x7f002e458ef0>
```



```
# Reshape matrices from 3D into 2D

X_sub_train = X_sub_train.reshape(X_sub_train.shape[0], (X_sub_train.shape[1]*X_sub_train.shape[2]))
X_sub_test = X_test.reshape(X_test.shape[0],(X_test.shape[1]*X_test.shape[2]))
```

```
# Change numerics into a matrix form for the target feature

n_classes = len(set(y_sub_train))
y_sub_train = to_categorical(y_sub_train, num_classes = n_classes)
y_sub_test = to_categorical(y_test, num_classes = n_classes)
```

```
# Split the dataset into the training set and test set

X_sub_train, X_sub_val, y_sub_train, y_sub_val = train_test_split(X_sub_train, y_sub_train, stratify = y_sub_train, tes
```

```
# Build a 2-layer dense neural network

model = Sequential([
    Dense(32, input_shape = (784,)),
    Activation("relu"),
    Dense(16),
    Activation("relu"),
    Dense(12),
    Activation("relu"),
    Dense(10),
    Activation("softmax")
])
```

```
# Configure the model for training

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

```
# Fit the model to the training data

model.fit(X_sub_train, y_sub_train, validation_split=0.3, batch_size = 32, epochs = 75, verbose = 0)
```

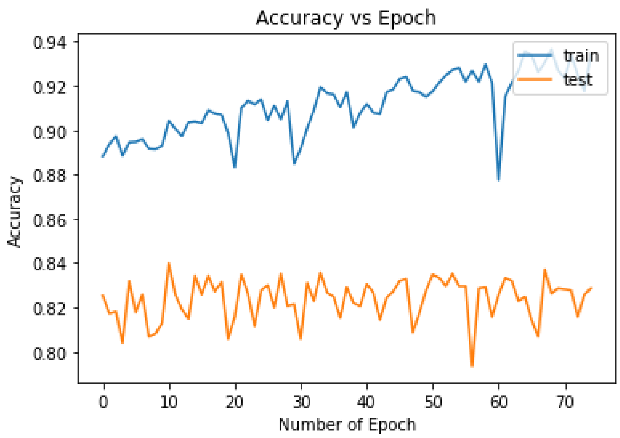
```
<keras.callbacks.History at 0x7f002e40dc88>
```

```
# Compute and print the test loss and accuracy

score = model.evaluate(X_sub_test, y_sub_test, verbose=0)
model1_loss = score[0]
model1_acc = score[1]
print("Test loss: {:.3f}".format(score[0]))
print("Test Accuracy: {:.3f}".format(score[1]))
```

```
Test loss: 0.834
Test Accuracy: 0.814
```

```
# Visualize a learning curve


epochs = 75

def learning_curve(model, X_train, y_train, epochs):


  model.compile(optimizer = "adam", loss="categorical_crossentropy", metrics=['accuracy'])
  model_hist = model.fit(X_train, y_train, validation_split=0.3, batch_size = 32, epochs = epochs, verbose = 0)

  plt.plot(model_hist.history['acc'])
  plt.plot(model_hist.history['val_acc'])
  plt.title("Accuracy vs Epoch")
  plt.ylabel("Accuracy")
  plt.xlabel("Number of Epoch")
  plt.xticks(np.arange(0, epochs+1, 10))
  plt.legend(['train', 'test'], loc='upper right')
  plt.show()

  return plt
```

```
learning_curve(model,X_sub_train, y_sub_train, epochs)
```




```
<module 'matplotlib.pyplot' from '/usr/local/lib/python3.6/dist-packages/matplotlib/pyplot.py'>
```

Vanilla model using drop-out

```
# Build the model using drop-out

from keras.layers import Dropout

dropout_model = Sequential ([
    Dense(100, input_shape = (784,), activation='relu'),
    Dropout(.25),
    Dense(50, activation='relu'),
    Dropout(.25),
    Dense(25, activation='relu'),
    Dropout(.25),
    Dense(10, activation='softmax'),

])
```

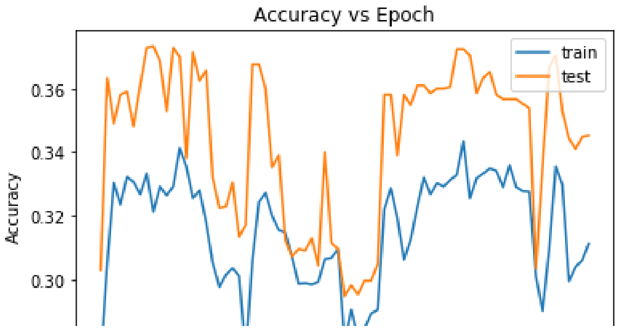
```
# Compile and fit the model to the data

dropout_model.compile(optimizer = "adam", loss="categorical_crossentropy",
                      metrics=['accuracy'])

dropout_model.fit(X_sub_train, y_sub_train, validation_split=0.3, batch_size = 32, epochs = 75, verbose = 0)
```

```
<keras.callbacks.History at 0x7f0012ef2d68>
```

```
# Compute and print the loss and accuracy

score = dropout_model.evaluate(X_sub_val, y_sub_val, verbose=0)
model2_loss = score[0]
model2_acc = score[1]
print("Test loss: {:.3f}".format(score[0]))
print("Test Accuracy: {:.3f}".format(score[1]))
```

```
Test loss: 11.610
Test Accuracy: 0.280
```

```
# Draw a learning curve

epochs = 75
learning_curve(dropout_model, X_sub_train, y_sub_train, epochs)
```



Model using batch normalization

```
# Data preprocessing

from keras.utils.np_utils import to_categorical

X_train_img = X_train.reshape(X_train.shape[0], X_train.shape[1], X_train.shape[2],1)
X_test_img = X_test.reshape(X_test.shape[0], X_test.shape[1], X_train.shape[2],1)


n_classes = len(set(y_train))
y_train_img = to_categorical(y_train, n_classes)
y_test_img = to_categorical(y_test, n_classes)
```

```
# Split the data into the training set and test set

X_train_bn, X_test_bn, y_train_bn, y_test_bn = train_test_split(X_train_img, y_train_img, stratify = y_train_img,
                                                               test_size = 0.3, random_state=11)
```

```
# Building the model with batch normaliation

from keras.layers import BatchNormalization
from keras.layers import Conv2D, MaxPooling2D, Flatten

input_shape = (28,28,1)

cnv_bn = Sequential([
    Conv2D(8, kernel_size = (3,3),
          input_shape = input_shape, activation = 'relu'),
    BatchNormalization(),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(8, kernel_size = (3,3),
          input_shape = (784,), activation = 'relu'),
    BatchNormalization(),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(32, activation='relu'),
    Dense(10, activation = 'softmax')

])
```

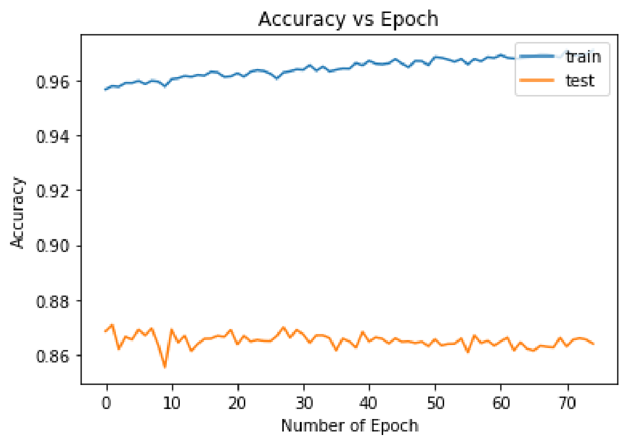
```
# Complile and fit the model to the data

cnv_bn.compile(optimizer = "adam", loss="categorical_crossentropy",
                      metrics=['accuracy'])

cnv_bn.fit(X_train_bn, y_train_bn, validation_split=0.3, batch_size = 32, epochs = 75, verbose = 0)
```

```
<keras.callbacks.History at 0x7f0013e230f0>
```

```
# Compute and print loss and accuracy

score = cnv_bn.evaluate(X_test_bn, y_test_bn, verbose=0)
model3_loss = score[0]
model3_acc = score[1]
print("Test loss: {:.3f}".format(score[0]))
print("Test Accuracy: {:.3f}".format(score[1]))
```

```
Test loss: 0.607
Test Accuracy: 0.870
```

```
# Draw a learning curve

epochs = 75
learning_curve(cnv_bn, X_train_bn, y_train_bn, epochs)
```




```
<module 'matplotlib.pyplot' from '/usr/local/lib/python3.6/dist-packages/matplotlib/pyplot.py'>
```

Model using residual connections without drop-out

```
# Data Preprocessing

X_train_rcnn = X_train.reshape(X_train.shape[0], (X_train.shape[1]* X_train.shape[2]))
X_test_rcnn = X_test.reshape(X_test.shape[0], (X_test.shape[1]* X_test.shape[2]))


n_classes = len(set(y_train))
y_train_rcnn = to_categorical(y_train, n_classes)
y_test_rcnn = to_categorical(y_test, n_classes)
```

```
# Split the data into the training and test set

X_train_rcnn_fit, X_test_rcnn_fit, y_train_rcnn_fit, y_test_rcnn_fit = train_test_split(X_train_rcnn, y_train_rcnn, str
                                                                                         test_size = 0.3, random_state=17
```

```
# Build the resnet model

from keras.layers import Input, Dense
from keras.models import Model

inputs = Input(shape=(784,))

x = Dense(64, activation='relu')(inputs)
BatchNormalization()
x = Dense(32, activation='relu')(x)
BatchNormalization()
x = Dense(16, activation='relu')(x)
BatchNormalization()
predictions = Dense(10, activation='softmax')(x)

rcnn_model = Model(inputs=inputs, outputs=predictions)
rcnn_model.compile(optimizer= 'adam',
              loss="categorical_crossentropy",
              metrics=['accuracy'])
```

```
rcnn_model.fit(X_sub_train, y_sub_train, batch_size = 32, epochs = 75, verbose = 0)
```

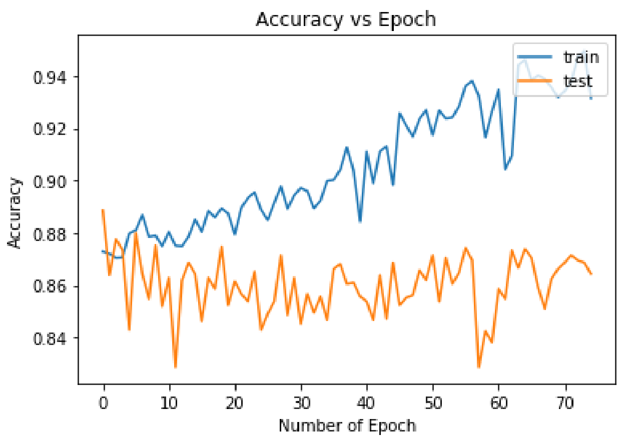
```
<keras.callbacks.History at 0x7f00125412e8>
```

```
#Compute and print the loss and accuracy
```

```
score = rcnn_model.evaluate(X_sub_test, y_sub_test, verbose=0)
model4_loss = score[0]
model4_acc = score[1]
print("Test loss: {:.3f}".format(score[0]))
print("Test Accuracy: {:.3f}".format(score[1]))
```

```
Test loss: 0.796
Test Accuracy: 0.799
```

```
# Draw a learning curve

epochs = 75
learning_curve(rcnn_model, X_sub_train, y_sub_train, epochs)
```




```
<module 'matplotlib.pyplot' from '/usr/local/lib/python3.6/dist-packages/matplotlib/pyplot.py'>
```

```
# Preparation for creating a dataframe for all the result

loss = {'loss':[model1_loss,model2_loss,model3_loss,model4_loss]}
acc = {'test accuracy':[model1_acc,model2_acc,model3_acc,model4_acc]}

loss_df = pd.DataFrame(data=loss,index = ['Vanila Model', 'Vanila Model Using Drop-out',
                                          'Batch Normalization', 'Residual Connections'])
acc_df = pd.DataFrame(data=acc, index = ['Vanila Model', 'Vanila Model Using Drop-out'
                                         , 'Batch Normalization', 'Residual Connections'])

df = df_concat = pd.concat([loss_df, acc_df], axis=1)
```

```
# Display the dataframe

df
```

	loss	test accuracy
Vanila Model	0.833648	0.814300
Vanila Model Using Drop-out	11.610402	0.279667
Batch Normalization	0.607016	0.870167
Residual Connections	0.796226	0.799000

Comparison: As we can obviously see, the bath normalization gives the lowest loss and the highest test accuaracy in the other models. The drop-out model performs the worst because it does not have the enogh number of hidden layer units but we set relatively high drop-out rate with respect to the number of hidden layer units. However, even though we increased more layers and set a lower drop-out rate; however, it got a higher loss and worse test accuracy score . The residual connections relatively produces the fair result. Since it is normally used when you have deeper networks but we have the small number of depth of our networks, it could have performed better. However, this task does not require to tune any parameters as well as the depth size. The vanila model also performs well. We could improve its accuracy, but the more layers we add, the more the model would tend to overfit.

Task 3.1

```
from google.colab import drive
drive.mount('/content/gdrive', force_remount=True)
```

```
Mounted at /content/gdrive
```

```
import zipfile, os
from scipy import misc
import imageio
import matplotlib.pyplot as pl
import os
import fnmatch
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from sklearn.model_selection import train_test_split
import cv2
from sklearn.model_selection import GridSearchCV, StratifiedShuffleSplit
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout
from sklearn.preprocessing import StandardScaler
```

```
Using TensorFlow backend.
```

```
import tensorflow as tf
device_name = tf.test.gpu_device_name()
if device_name != '/device:GPU:0':
    raise SystemError('GPU device not found')
```

```
Found GPU at: /device:GPU:0
```

```
# UNZIP ZIP
# print ("Uncompressing zip file")
# zip_ref = zipfile.ZipFile('/content/gdrive/My Drive/Columbia/Spring2019/aml/hw5/b
# zip_ref.extractall('/content/gdrive/My Drive/Columbia/Spring2019/aml/hw5/')
```

```
Uncompressing zip file
```

```
# print ("Uncompressing zip file")
# zip_ref = zipfile.ZipFile('/content/gdrive/My Drive/Columbia/Spring2019/aml/hw5/I
# zip_ref.extractall('/content/gdrive/My Drive/Columbia/Spring2019/aml/hw5/dataset/
```

```
Uncompressing zip file
```

```
breast-histopathology-images.zip  IDC_regular_ps50_idx5.zip  New
dataset                           new                        task3.ipynb
```