

Lab Report – 2

Goutham Krishnan

21BAI1007

Aim

1. Perform classification on the PIMA diabetes dataset using K folds and Grid search autotuning.
2. Perform MNIST dataset classification using K Folds and Grid Search autotuning.
3. Perform Fashion MNIST classification using K Folds and Grid Search autotuning.
4. Perform classification on the Cats and Dogs dataset using K folds and Grid search autotuning.
5. Perform classification on the PCAM dataset using K folds and Grid Search autotuning.

Code and Output

Sample Code on K folds and Grid Search

The screenshot shows a Google Colab notebook titled "21BAI1007_KFold_Sample.ipynb". The code cell contains Python code for performing K-fold cross-validation on the PIMA diabetes dataset. The code imports necessary libraries (keras, sklearn), sets a seed, loads the dataset, splits it into training and testing sets, creates a sequential model with two hidden layers, compiles it with binary_crossentropy loss and Adam optimizer, fits it to the training data, evaluates it on the test data, and prints the accuracy. The output shows multiple runs of the loop with varying accuracies (e.g., 70.22%, 71.43%, 80.52%, 88.52%). A warning message about tensor retracing is visible at the bottom.

```
[ ] from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import StratifiedKFold
import numpy

[ ] seed = 7
numpy.random.seed(seed)
dataset = numpy.genfromtxt("./content/diabetes.csv", delimiter=",", skip_header=1)

[ ] X = dataset[:, 0:8]
Y = dataset[:, 8]

[ ] kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state = seed)
cvscores = []

[ ] for train, test in kfold.split(X, Y):
    model = Sequential()
    model.add(Dense(2, input_dim = 8, kernel_initializer='uniform', activation="relu"))
    model.add(Dense(8, kernel_initializer='uniform', activation = "relu"))
    model.add(Dense(1, kernel_initializer='uniform', activation = "sigmoid"))
    model.compile(loss="binary_crossentropy", optimizer="adam", metrics = ["accuracy"])
    model.fit(X[train], Y[train], epochs = 150, batch_size = 10, verbose=0 )
    scores = model.evaluate(X[test], Y[test], verbose = 0)
    print("Xs: %.2f%% (%s) %s" % (model.metrics_names[1], scores[1]*100))
    cvscores.append(scores[1] * 100)

accuracy: 70.22%
accuracy: 79.22%
accuracy: 71.43%
accuracy: 80.52%
accuracy: 88.52%
WARNING:tensorflow:5 out of the last 13 calls to <function Model.make_test_function.<locals>.test_function at 0x7935d8fc550> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to ...
accuracy: 71.43%
WARNING:tensorflow:5 out of the last 13 calls to <function Model.make_test_function.<locals>.test_function at 0x7935d800a6000> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to ...
accuracy: 61.64%
accuracy: 76.62%
accuracy: 79.22%
accuracy: 75.06%
accuracy: 77.63%
```

The screenshot shows a Google Colab notebook titled "21BAI1007_KFold_Sample.ipynb". The code cell contains Python code for performing GridSearchCV on the PIMA diabetes dataset. It prints the mean and standard deviation of the cross-validation scores, installs scikeras, and then imports necessary libraries (keras, scikeras, sklearn) and performs a grid search on a sequential model with two hidden layers and a KerasClassifier.

```
[ ] print("%.2f%% (+/- %.2f%%)" % (numpy.mean(cvscores), numpy.std(cvscores)))

74.35% (+/- 5.14%)

{ } Sample 2 - GridSearch - 21BAI1007

[ ] pip install scikeras

Collecting scikeras
  Downloading scikeras-0.12.0-py3-none-any.whl (27 kB)
Requirement already satisfied: packaging>=0.21 in /usr/local/lib/python3.10/dist-packages (from scikeras) (23.2)
Requirement already satisfied: scikit-learn>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from scikeras) (1.2.2)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0>scikeras) (1.23.5)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0>scikeras) (1.11.4)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0>scikeras) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0>scikeras) (3.2.0)
Installing collected packages: scikeras
Successfully installed scikeras-0.12.0

[ ] from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Wrapper
from scikeras.wrappers import KerasClassifier
from sklearn.model_selection import GridSearchCV
import numpy
```

21BAI1007_KFold_Sample.ipynb

```
[ ] def create_model(optimizer="rmsprop", init = "glorot_uniform"):
    model = Sequential()
    model.add(Dense(12, input_dim = 8, kernel_initializer=init, activation = "relu"))
    model.add(Dense(8, kernel_initializer = init, activation="relu"))
    model.add(Dense(1, kernel_initializer=init, activation="sigmoid"))
    model.compile(loss="binary_crossentropy", optimizer=optimizer, metrics=["accuracy"])
    return model

[ ] seed = 7
numpy.random.seed(seed)
dataset = numpy.loadtxt("/content/diabetes.csv", delimiter=",", skip_header=1)

[ ] X=dataset[:, 0:8]
Y=dataset[:, 8]

[ ] model = KerasClassifier(model=create_model, verbose=0)

optimizers = ["rmsprop", "adam"]
epochs = [50, 100, 150]
batches = [5, 10, 20]
param_grid = dict(optimizer=optimizers, epochs=epochs, batch_size=batches)

grid = GridSearchCV(estimator=model, param_grid=param_grid)
grid_result = grid.fit(X, Y)

[ ] # summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))

Best: 0.726670 using {'batch_size': 5, 'epochs': 150, 'optimizer': 'rmsprop'}
```

Start coding or generate with AI.

PIMA Dataset classification using autotuning

pima-21BAI1007.ipynb

```
[ ] import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

[ ] from google.colab import drive
drive.mount('/content/drive')

Mounted at /content/drive

[ ] df = pd.read_csv('/content/drive/MyDrive/Datasets/diabetes.csv')
df.head()
```

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

pima-21BAI1007.ipynb

```
[ ] display(df.info(),df.head())

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype  
 --- 
 0   Pregnancies      768 non-null    int64  
 1   Glucose          768 non-null    int64  
 2   BloodPressure    768 non-null    int64  
 3   SkinThickness    768 non-null    int64  
 4   Insulin          768 non-null    int64  
 5   BMI              768 non-null    float64 
 6   DiabetesPedigreeFunction 768 non-null    float64 
 7   Age              768 non-null    int64  
 8   Outcome          768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
None

Pregnancies Glucose BloodPressure SkinThickness Insulin BMI DiabetesPedigreeFunction Age Outcome
0            6       148            72          35     0  33.6        0.627  50       1
1            1        85            66          29     0  26.6        0.351  31       0
2            8       183            64          0     0  23.3        0.672  32       1
3            1        89            66          23     94  28.1        0.167  21       0
4            0       137            40          35     168  43.1        2.288  33       1
```

pima-21BAI1007.ipynb

```
[{x}] # Checking for null values if any
df.isna().sum()

{x} Pregnancies      0
     Glucose        0
     BloodPressure   0
     SkinThickness   0
     Insulin         0
     BMI             0
     DiabetesPedigreeFunction 0
     Age             0
     Outcome         0
     dtype: int64

[ ] ## Extracting the features
x = df.iloc[:, :-1]
y = df.iloc[:, -1:]

[ ] x.head()

Pregnancies Glucose BloodPressure SkinThickness Insulin BMI DiabetesPedigreeFunction Age
0            6       148          72           35      0  33.6      0.627  50
1            1        85          66           29      0  26.6      0.351  31
2            8       183          64           0      0  23.3      0.672  32
3            1        89          66           23      94  28.1      0.167  21
4            0       137          40           35      168 43.1      2.288  33
```

pima-21BAI1007.ipynb

```
+ Code + Text
Creating the model - 21BAI1007

{x}
[ ] from sklearn.model_selection import StratifiedKFold
kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=7)
cvscores = []

[ ] from keras.models import Sequential
from keras.layers import Dense, Dropout

[ ] for train, test in kfold.split(x, y):

    model = Sequential()
    model.add(Dense(512, input_dim=8, kernel_initializer='uniform', activation='relu'))
    model.add(Dense(256, kernel_initializer='uniform'))
    model.add(Dense(128, kernel_initializer='uniform'))
    model.add(Dense(32))
    model.add(Dense(1, activation='sigmoid'))

    model.compile(loss="binary_crossentropy", optimizer="adam", metrics=[ "accuracy" ])
    # fit the model
    model.fit(x[train], y[train], epochs=150, verbose=0) # evaluate the model
    scores = model.evaluate(x[test], y[test], verbose=0)
    print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
    cvscores.append(scores[1] * 100)

accuracy: 75.32%
accuracy: 75.32%
accuracy: 71.43%
accuracy: 79.22%
WARNING:tensorflow:5 out of the last 13 calls to <function Model.make_test_function.<locals>.test_function at 0x7ad4307bab00> triggered tf.function retracing. Tracing is expensive and the excessive number of
accuracy: 68.83%
WARNING:tensorflow:5 out of the last 13 calls to <function Model.make_test_function.<locals>.test_function at 0x7ad43031c820> triggered tf.function retracing. Tracing is expensive and the excessive number of
accuracy: 67.53%
accuracy: 76.02%
accuracy: 75.32%
```

pima-21BAI1007.ipynb

```
+ Code + Text
[ ] accuracy: 75.32%
accuracy: 75.32%
accuracy: 71.43%
accuracy: 79.22%
WARNING:tensorflow:5 out of the last 13 calls to <function Model.make_test_function.<locals>.test_function at 0x7ad4307bab00> triggered tf.function retracing. Tracing is expensive and the excessive number of
accuracy: 68.83%
WARNING:tensorflow:5 out of the last 13 calls to <function Model.make_test_function.<locals>.test_function at 0x7ad43031c820> triggered tf.function retracing. Tracing is expensive and the excessive number of
accuracy: 67.53%
accuracy: 76.02%
accuracy: 75.32%
accuracy: 77.63%
accuracy: 82.89%
[ ] print("%.2f%% (+/- %.2f%%)" % (np.mean(cvscores), np.std(cvscores)))
75.01% (+/- 4.43%)

Predicting the test output - 21BAI1007

[ ] probs = model.predict(x[test])
3/3 [=====] - 0s 3ms/step

[ ] th = 0.5
y_pred = np.where(probs > th, 1, 0)

[ ] from sklearn.metrics import accuracy_score
print(accuracy_score(np.array(y[test]), y_pred)*100, "%")
82.89473684210526 %
```

MNIST Dataset Classification using Autotuning

```
[ ] from keras.datasets import mnist
import numpy as np

[ ] (train_images, train_labels), (test_images, test_labels)= mnist.load_data()

[ ] train_images.shape, train_labels.shape
((60000, 28, 28), (60000,))

[ ] len(train_labels), len(test_labels)
(60000, 10000)

[ ] from keras import models
from keras import layers

[ ] train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype('float32') / 255
test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype('float32') / 255

[ ] train_labels
array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)
```

```
[ ] features = np.concatenate((train_images, test_images))
labels = np.concatenate((train_labels, test_labels))

[ ] labels
array([5, 0, 4, ..., 4, 5, 6], dtype=uint8)

[ ] Creating the model using K fold autotuning - 21BAI1007

[ ] from sklearn.model_selection import StratifiedKFold
kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=7)
cvscores = []

[ ] from keras.models import Sequential
from keras.layers import Dense, Flatten

for train, test in kfold.split(features, labels):

    model = Sequential()
    model.add(Dense(512, activation='relu', input_shape=(28*28,)))
    model.add(Dense(10, activation='softmax'))

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

    model.fit(features[train], labels[train], epochs=10, batch_size=128, validation_data=(features[test], labels[test])) # evaluate the model
    scores = model.evaluate(features[test], labels[test], verbose=1)
    print("%s: %d" % (model.metrics_names[1], scores[1]))
    cvscores.append(scores[1] * 100)
```

```
[ ] 438/438 [=====] - 2s 5ms/step - loss: 0.0718 - accuracy: 0.9786 - val_loss: 0.0798 - val_accuracy: 0.9752
Epoch 4/10
438/438 [=====] - 2s 4ms/step - loss: 0.0498 - accuracy: 0.9854 - val_loss: 0.0721 - val_accuracy: 0.9775
Epoch 5/10
438/438 [=====] - 1s 3ms/step - loss: 0.0370 - accuracy: 0.9893 - val_loss: 0.0676 - val_accuracy: 0.9784
Epoch 6/10
438/438 [=====] - 1s 3ms/step - loss: 0.0276 - accuracy: 0.9921 - val_loss: 0.0728 - val_accuracy: 0.9784
Epoch 7/10
438/438 [=====] - 2s 4ms/step - loss: 0.0207 - accuracy: 0.9945 - val_loss: 0.0639 - val_accuracy: 0.9821
Epoch 8/10
438/438 [=====] - 2s 3ms/step - loss: 0.0150 - accuracy: 0.9962 - val_loss: 0.0635 - val_accuracy: 0.9819
Epoch 9/10
438/438 [=====] - 2s 3ms/step - loss: 0.0109 - accuracy: 0.9976 - val_loss: 0.0660 - val_accuracy: 0.9821
Epoch 10/10
438/438 [=====] - 1s 3ms/step - loss: 0.0092 - accuracy: 0.9978 - val_loss: 0.0678 - val_accuracy: 0.9816
accuracy: 0
Epoch 1/10
438/438 [=====] - 2s 4ms/step - loss: 0.2709 - accuracy: 0.9230 - val_loss: 0.1536 - val_accuracy: 0.9579
Epoch 2/10
438/438 [=====] - 2s 3ms/step - loss: 0.1084 - accuracy: 0.9699 - val_loss: 0.1095 - val_accuracy: 0.9685
Epoch 3/10
438/438 [=====] - 2s 4ms/step - loss: 0.0707 - accuracy: 0.9794 - val_loss: 0.0901 - val_accuracy: 0.9739
Epoch 4/10
438/438 [=====] - 3s 6ms/step - loss: 0.0499 - accuracy: 0.9855 - val_loss: 0.0845 - val_accuracy: 0.9757
Epoch 5/10
438/438 [=====] - 2s 4ms/step - loss: 0.0375 - accuracy: 0.9889 - val_loss: 0.0830 - val_accuracy: 0.9759
Epoch 6/10
438/438 [=====] - 1s 3ms/step - loss: 0.0276 - accuracy: 0.9929 - val_loss: 0.0812 - val_accuracy: 0.9770
Epoch 7/10
438/438 [=====] - 1s 3ms/step - loss: 0.0196 - accuracy: 0.9947 - val_loss: 0.0898 - val_accuracy: 0.9740
Epoch 8/10
438/438 [=====] - 1s 3ms/step - loss: 0.0152 - accuracy: 0.9962 - val_loss: 0.0766 - val_accuracy: 0.9797
Epoch 9/10
438/438 [=====] - 1s 3ms/step - loss: 0.0114 - accuracy: 0.9975 - val_loss: 0.0825 - val_accuracy: 0.9782
Epoch 10/10
438/438 [=====] - 1s 3ms/step - loss: 0.0102 - accuracy: 0.9976 - val_loss: 0.0802 - val_accuracy: 0.9788
accuracy: 0
```

```
# pip install scikeras
# from keras.layers import Wrapper
# from scikeras.wrappers import KerasClassifier
# from sklearn.model_selection import GridSearchCV

# def create_model(optimizer="rmsprop", init = "glorot_uniform"):
#     model = Sequential()
#     model.add(Flatten(input_shape=(28,28)))
#     model.add(Dense(128, activation='relu'))
#     model.add(Dense(10, activation='softmax'))
#     model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics='accuracy')
#     return model

# model2 = KerasClassifier(model=create_model, verbose=0)

# optimizers = ["rmsprop", "adam"]
# epochs = [50, 100]
# batches = [5, 10, 20]
# param_grid = dict(optimizer=optimizers, epochs=epochs, batch_size=batches)

# grid = GridSearchCV(estimator=model2, param_grid=param_grid)
# grid_result = grid.fit(features, labels)

# ** Grid search is taking a lot of time to run, hence not including the output **
```

```
[ ] # ** Grid search is taking a lot of time to run, hence not including the output **

{ } Predicting the labels for the test images - 21BAI1007

[ ] pred = model.predict(features[test])
438/438 [=====] - 1s/step

[ ] predicted_labels = np.argmax(pred, axis=1)
print(predicted_labels)

[3 5 3 ... 2 1 2]

[ ] labels[test]
array([3, 5, 3, ..., 2, 1, 2], dtype=uint8)

[ ] from sklearn.metrics import accuracy_score
print("Accuracy: ",accuracy_score(predicted_labels, labels[test])*100, "%")

Accuracy: 97.87857142857142 %
```

Fashion MNIST dataset classification using autotuning

```
[ ] import numpy as np

[ ] from keras.datasets import fashion_mnist
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
29515/29515 [=====] - 0s/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26421880/26421880 [=====] - 0s/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
5148/5148 [=====] - 0s/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 [=====] - 0s/step

[ ] print("Shape of training dataset:",train_images.shape)
print("Shape of training labels:",train_labels.shape)

Shape of training dataset: (60000, 28, 28)
Shape of training labels: (60000,)

We have 60000 images, each of size 28*28 pixels. Each label is an integer between 0 and 9

[ ] print("Length of training dataset: ",len(train_images))
print("Length of testing dataset: ",len(test_images))

Length of training dataset: 60000
Length of testing dataset: 10000
```

We have 60000 images, each of size 28*28 pixels. Each label is an integer between 0 and 9

```
[ ] print("Length of training dataset: ",len(train_images))
print("Length of testing dataset: ",len(test_images))

Length of training dataset: 60000
Length of testing dataset: 10000
```

Normalizing the dataset - 21BAI1007

```
[ ] import matplotlib.pyplot as plt
[ ] ## Plotting the image of the first object in the dataset
plt.figure()
plt.imshow(train_images[0])
```

```
[ ] ## To normalize the dataset, we divide each value by 255 so that the data is minimized to a value between 0 and 1
train_images = train_images/255.0
test_images = test_images/255.0
```

Build the model using K Fold auto tuning- 21BAI1007

```
[ ] features = np.concatenate((train_images, test_images))
labels = np.concatenate((train_labels, test_labels))

[ ] from keras.models import Sequential
from keras.layers import Dense, Flatten

[ ] from sklearn.model_selection import StratifiedKFold
kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=7)
cvscores = []
```

```
[ ] for train, test in kfold.split(features, labels):
    model = Sequential()
    model.add(Flatten(input_shape=(28,28)))
    model.add(Dense(128, activation='relu'))
    model.add(Dense(10, activation='softmax'))
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics='accuracy')
    model.fit(features[train], labels[train], validation_data=(features[test], labels[test]), epochs=10)
    scores = model.evaluate(features[test], labels[test], verbose=1)
    print("%s: %d" % (model.metrics_names[1], scores[1]))
    cvscores.append(scores[1] * 100)

Epoch 1/10
1750/1750 [=====] - 6s 3ms/step - loss: 0.5051 - accuracy: 0.8213 - val_loss: 0.4371 - val_accuracy: 0.8414
Epoch 2/10
1750/1750 [=====] - 6s 3ms/step - loss: 0.3822 - accuracy: 0.8621 - val_loss: 0.3690 - val_accuracy: 0.8704
Epoch 3/10
1750/1750 [=====] - 5s 3ms/step - loss: 0.3413 - accuracy: 0.8752 - val_loss: 0.3557 - val_accuracy: 0.8739
Epoch 4/10
1750/1750 [=====] - 6s 3ms/step - loss: 0.3162 - accuracy: 0.8833 - val_loss: 0.3426 - val_accuracy: 0.8788
Epoch 5/10
1750/1750 [=====] - 6s 3ms/step - loss: 0.3008 - accuracy: 0.8875 - val_loss: 0.3243 - val_accuracy: 0.8862
Epoch 6/10
1750/1750 [=====] - 5s 3ms/step - loss: 0.2842 - accuracy: 0.8934 - val_loss: 0.3135 - val_accuracy: 0.8928
Epoch 7/10
1750/1750 [=====] - 6s 4ms/step - loss: 0.2715 - accuracy: 0.8985 - val_loss: 0.3690 - val_accuracy: 0.8781
Epoch 8/10
1750/1750 [=====] - 5s 3ms/step - loss: 0.2602 - accuracy: 0.9020 - val_loss: 0.3179 - val_accuracy: 0.8930
Epoch 9/10
1750/1750 [=====] - 6s 3ms/step - loss: 0.2501 - accuracy: 0.9057 - val_loss: 0.3292 - val_accuracy: 0.8851
Epoch 10/10
1750/1750 [=====] - 5s 3ms/step - loss: 0.2413 - accuracy: 0.9088 - val_loss: 0.3057 - val_accuracy: 0.8937
438/438 [=====] - 1s 2ms/step - loss: 0.3057 - accuracy: 0.8937
accuracy: 0
Epoch 1/10
1750/1750 [=====] - 7s 3ms/step - loss: 0.5110 - accuracy: 0.8207 - val_loss: 0.4377 - val_accuracy: 0.8391
Epoch 2/10
1750/1750 [=====] - 5s 3ms/step - loss: 0.3892 - accuracy: 0.8602 - val_loss: 0.3626 - val_accuracy: 0.8702
Epoch 3/10
1750/1750 [=====] - 3s 2ms/step - loss: 0.3460 - accuracy: 0.8933 - val_loss: 0.3630 - val_accuracy: 0.8683
```

fashionmnist_21BAI1007.ipynb

File Edit View Insert Runtime Tools Help Last saved at 21:41

Comment Share G

Connect T4 | Colab AI

```
+ Code + Text
[ ] Epoch 1/10
  1750/1750 [=====] - 7s 3ms/step - loss: 0.5112 - accuracy: 0.8203 - val_loss: 0.4079 - val_accuracy: 0.8551
Epoch 2/10
  1750/1750 [=====] - 5s 3ms/step - loss: 0.3818 - accuracy: 0.8626 - val_loss: 0.3761 - val_accuracy: 0.8627
Epoch 3/10
  1750/1750 [=====] - 5s 3ms/step - loss: 0.3424 - accuracy: 0.8767 - val_loss: 0.3324 - val_accuracy: 0.8795
Epoch 4/10
  1750/1750 [=====] - 6s 3ms/step - loss: 0.3194 - accuracy: 0.8812 - val_loss: 0.3373 - val_accuracy: 0.8769
Epoch 5/10
Fnach 5/10
```

Creating model using grid search - 21BAI1007

```
[ ] # pip install scikeras
# from keras.layers import Wrapper
# from scikeras.wrappers import KerasClassifier
# from sklearn.model_selection import GridSearchCV

# def create_model(optimizer='rmsprop', init = "glorot_uniform"):
#     model = Sequential()
#     model.add(Flatten(input_shape=(28,28)))
#     model.add(Dense(128, activation='relu'))
#     model.add(Dense(10, activation='softmax'))
#     model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics='accuracy')
#     return model

# model2 = KerasClassifier(create_model, verbose=0)

# optimizers = ["rmsprop", "adam"]
# epochs = [50, 100]
# batches = [5, 10, 20]
# param_grid = dict(optimizer=optimizers, epochs=epochs, batch_size=batches)

# grid = GridSearchCV(estimator=model2, param_grid=param_grid)
# grid_result = grid.fit(features, labels)

# ** Grid search is taking a lot of time to run, hence not including the output **
```

fashionmnist_21BAI1007.ipynb

File Edit View Insert Runtime Tools Help Last saved at 21:41

Comment Share G

Connect T4 | Colab AI

Predicting the classes of the test dataset - 21BAI1007

```
[ ] pred = model.predict(test_images)
313/313 [=====] - 1s 1ms/step

[ ] # The predictions are a probability distribution of each class. To convert it into the required class
# Predicting the class of the first element in the testing dataset
np.argmax(pred[0])
9

[ ] # Checking if the prediction is correct
test_labels[0]
9

[ ] class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

Prediciting a random image from the test dataset - 21BAI1007

```
[ ] image = test_images[1] ## An image of a pullover
print(image.shape)
(28, 28)
```

fashionmnist_21BAI1007.ipynb

File Edit View Insert Runtime Tools Help Last saved at 21:41

Comment Share G

Connect T4 | Colab AI

+ Code + Text

```
[ ] plt.imshow(test_images[1])
<matplotlib.image.AxesImage at 0x791c4038a920>
```

```
[ ] # To process the image in the model, we have to reshape it no include an extra dimension
image = (np.expand_dims(image, 0))
print(image.shape)
(1, 28, 28)
```

```

[ ] # To process the image in the model, we have to reshape it no include an extra dimension
image = (np.expand_dims(image, 0))
print(image.shape)

(1, 28, 28)

[ ] # Function to map the class number to the class name
def get_item_name(item_number, class_names):
    try:
        item_name = class_names[item_number]
        return item_name
    except IndexError:
        return "Item not found"

[ ] final_prediction = model.predict(image)
ans = np.argmax(final_prediction)
item_name = get_item_name(ans, class_names)
item_name

1/1 [=====] - 0s 69ms/step
'Pullover'

Hence we correctly predicted the class of the image

```

Cats Vs Dogs Dataset Classification using Autotuning

```

[ ] mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle

[ ] !kaggle datasets download -d salader/dogs-vs-cats

Warning: Your Kaggle API key is readable by other users on this system! To fix this, you can run 'chmod 600 /root/.kaggle/kaggle.json'
Downloaded dogs-vs-cats.zip to /content
99% 1.05G/1.06G [00:07<00:00, 93.0MB/s]
100% 1.06G/1.06G [00:07<00:00, 15.9MB/s]

[ ] # Data loaded from kaggle is in zip format. Need to unzip it
import zipfile
zip_ref = zipfile.ZipFile('/content/dogs-vs-cats.zip', 'r')
zip_ref.extractall('/content')
zip_ref.close()

```

```

[ ] import tensorflow as tf
import keras
from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPooling2D, Flatten, BatchNormalization, Dropout

[ ] train_cats_path = '/content/train/cats'
train_dogs_path = '/content/train/dogs'
test_cats_path = '/content/test/cats'
test_dogs_path = '/content/test/dogs'

[ ] import os
def load_and_preprocess_images(directory, label, target_size=(64, 64)):

    x = []
    y = []

    for filename in os.listdir(directory):
        if filename.endswith('.jpg') or filename.endswith('.png'):
            filepath = os.path.join(directory, filename)
            img = cv2.imread(filepath)
            if img is not None:
                img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
                img = cv2.resize(img, target_size)
            x.append(img)
            y.append(label)

    return x,y

```

CatsDogs_KFold_21BAI1007.ipynb

```
+ Code + Text
import cv2
import numpy as np
X_cats_train, Y_cats_train = load_and_preprocess_images(train_cats_path, label=0)
X_dogs_train, Y_dogs_train = load_and_preprocess_images(train_dogs_path, label=1)
x_train = X_cats_train + X_dogs_train
y_train = Y_cats_train + Y_dogs_train
X_train = np.array(x_train)
Y_train = np.array(y_train)
X_cats_test, Y_cats_test = load_and_preprocess_images(test_cats_path, label=0)
X_dogs_test, Y_dogs_test = load_and_preprocess_images(test_dogs_path, label=1)
x_test = X_cats_test + X_dogs_test
y_test = Y_cats_test + Y_dogs_test
X_test = np.array(x_test)
Y_test = np.array(y_test)

[ ] X_train.shape
(20000, 64, 64, 3)
```

CatsDogs_KFold_21BAI1007.ipynb

```
+ Code + Text
Creating the model using K Folds Method - 21BAI1007
+ Code + Text
import tensorflow as tf
from tensorflow.keras import layers, models
from sklearn.model_selection import StratifiedKFold

kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

cvscores = []
for train_idx, test_idx in kfold.split(X_train, Y_train):

    model = models.Sequential()
    model.add(layers.Conv2D(32, kernel_size=(3, 3), padding='valid', activation='relu', input_shape=(64, 64, 3)))
    model.add(layers.BatchNormalization())
    model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=2, padding='valid'))
    model.add(layers.Flatten())
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dropout(0.1))
    model.add(layers.Dense(1, activation='sigmoid'))
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

    model.fit(X_train[train_idx], Y_train[train_idx], epochs = 5)

    scores = model.evaluate(X_test, Y_test, verbose = 0)
    print("%s: %."2f%%" % (model.metrics_names[1], scores[1] * 100))
    cvscores.append(scores[1] * 100)
```

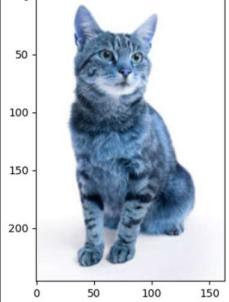
CatsDogs_KFold_21BAI1007.ipynb

```
+ Code + Text
[ ] epoch 2/5
500/500 [=====] - 2s 5ms/step - loss: 0.4981 - accuracy: 0.7492
Epoch 3/5
500/500 [=====] - 2s 5ms/step - loss: 0.4065 - accuracy: 0.8048
Epoch 4/5
500/500 [=====] - 2s 4ms/step - loss: 0.3134 - accuracy: 0.8563
Epoch 5/5
500/500 [=====] - 2s 4ms/step - loss: 0.2352 - accuracy: 0.8896
accuracy: 71.70%
Epoch 1/5
500/500 [=====] - 3s 4ms/step - loss: 0.7349 - accuracy: 0.6544
Epoch 2/5
500/500 [=====] - 3s 5ms/step - loss: 0.4902 - accuracy: 0.7450
Epoch 3/5
500/500 [=====] - 2s 4ms/step - loss: 0.3924 - accuracy: 0.8930
Epoch 4/5
500/500 [=====] - 2s 4ms/step - loss: 0.3314 - accuracy: 0.8426
Epoch 5/5
500/500 [=====] - 2s 4ms/step - loss: 0.2759 - accuracy: 0.8706
accuracy: 65.70%
Epoch 1/5
500/500 [=====] - 3s 4ms/step - loss: 0.7066 - accuracy: 0.6507
Epoch 2/5
500/500 [=====] - 2s 4ms/step - loss: 0.4920 - accuracy: 0.7429
Epoch 3/5
500/500 [=====] - 3s 5ms/step - loss: 0.4140 - accuracy: 0.7899
Epoch 4/5
500/500 [=====] - 2s 4ms/step - loss: 0.3424 - accuracy: 0.8345
Epoch 5/5
500/500 [=====] - 2s 4ms/step - loss: 0.2848 - accuracy: 0.8702
accuracy: 71.84%
Epoch 1/5
500/500 [=====] - 3s 4ms/step - loss: 0.7351 - accuracy: 0.6616
Epoch 2/5
500/500 [=====] - 2s 4ms/step - loss: 0.4760 - accuracy: 0.7658
Epoch 3/5
500/500 [=====] - 3s 5ms/step - loss: 0.3742 - accuracy: 0.8231
Epoch 4/5
500/500 [=====] - 2s 4ms/step - loss: 0.2944 - accuracy: 0.8651
Epoch 5/5
```

CatsDogs_KFold_21BAI1007.ipynb

```
[ ] import cv2
import matplotlib.pyplot as plt
test_image = cv2.imread('/content/cat.jpg')

[ ] plt.imshow(test_image)
<matplotlib.image.AxesImage at 0x7b5b625e90c0>
```



CatsDogs_KFold_21BAI1007.ipynb

```
[ ] test_image = cv2.resize(test_image,(64, 64))
test_input = test_image.reshape(1,64, 64,3)

[ ] predictions_probs = model.predict(test_input)
predictions = np.argmax(predictions_probs)
predictions

1/1 [=====] - 0s 18ms/step
0

Array[0] > Cat
Hence, the model has correctly predicted the given image of a cat
```

CatsDogs_KFold_21BAI1007.ipynb

```
Using the model correctly predicted the given image of a cat
```

Creating model using grid search - 21BAI1007

** Grid search is taking a lot of time to run, hence not including the output **

```
[ ] # pip install scikeras

# from keras.layers import Wrapper
# from scikeras.wrappers import KerasClassifier
# from sklearn.model_selection import GridSearchCV

# def create_model(optimizer="rmsprop", init = "glorot_uniform"):
#     model = models.Sequential()
#     model.add(layers.Conv2D(32, kernel_size=(3, 3), padding='valid', activation='relu', input_shape=(64, 64, 3)))
#     model.add(layers.BatchNormalization())
#     model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=2, padding='valid'))
#     model.add(layers.Flatten())
#     model.add(layers.Dense(64, activation='relu'))
#     model.add(layers.Dropout(0.1))
#     model.add(layers.Dense(1, activation='sigmoid'))
#     return model

# model2 = KerasClassifier(model=create_model, verbose=0)

# optimizers = ["rmsprop", "adam"]
# epochs = [50, 100]
# batches = [5, 10, 20]
# param_grid = dict(optimizer=optimizers, epochs=epochs, batch_size=batches)

# grid = GridSearchCV(estimator=model2, param_grid=param_grid)
# grid_result = grid.fit(X_train, Y_train)
```

PCAM Dataset Classification using Autotuning.

21BAI1007_PCAM_KFolds.ipynb

File Edit View Insert Runtime Tools Help Last saved at 21:40

+ Code + Text

PCAM Dataset Classification using K Folds autotuning

{x} 21BAI1007 - Goutham Krishnan

```
[ ] import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os
from sklearn.model_selection import train_test_split
from keras.optimizers import RMSprop, SGD, Adam
from keras.preprocessing.image import ImageDataGenerator
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Activation
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import OneHotEncoder
import numpy as np
import cv2
```

>Loading the data - 21BAI1007

```
[ ] !mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
```

```
[ ] !kaggle competitions download -c histopathologic-cancer-detection
```

```
Warning: Your Kaggle API key is readable by other users on this system! To fix this, you can run 'chmod 600 /root/.kaggle/kaggle.json'
Downloaded histopathologic-cancer-detection.zip to /content
100% 6.31G/6.31G [04:53<00:00, 25.4MB/s]
100% 6.31G/6.31G [04:53<00:00, 23.1MB/s]
```

21BAI1007_PCAM_KFolds.ipynb

File Edit View Insert Runtime Tools Help Last saved at 21:40

+ Code + Text

```
[ ] import zipfile
zip_ref = zipfile.ZipFile('/content/histopathologic-cancer-detection.zip', 'r')
zip_ref.extractall('/content')
zip_ref.close()
```

```
[ ] df=pd.read_csv('/content/train_labels.csv')
df
```

	id	label
0	f38a6374c348f90b587e046aac6079959ad3835	0
1	c18f2d887b7ae4f6742ee445113fa1afef383ed77	1
2	755db6279da599eb4d39a9123cce439965282d	0
3	b3c10c64fb68f4a8bd33a6f971ecae77c75e08	0
4	068aba587a4950175d04c680d38943d488d6a9d	0
...
220020	53e9aa9d46720bf3c6a7528d1fca3ba6e2e49f6	0
220021	d4b854fe38b07e2831ad73892b5cce877689576	1
220022	3d046cead1a2a5cbe00b2b4847cfb7ba7cf5fe75	0
220023	f12969113433f661e0671ff1f80944816fa2	0
220024	a81f84895ddcd522302ddf34be02eb1b3e5af1cb	1

220025 rows × 2 columns

21BAI1007_PCAM_KFolds.ipynb

File Edit View Insert Runtime Tools Help Last saved at 21:40

+ Code + Text

Preprocessing the data - 21BAI1007

```
[x] df["path"] = df["id"].apply(lambda x: os.path.join("/content/train", str(x) + ".tif"))
```

```
[ ] df["label"] = df["label"].astype(str)
df_0 = df[df["label"] == "0"].sample(10000, random_state=42)
df_1 = df[df["label"] == "1"].sample(10000, random_state=42)
df_subset = pd.concat([df_0, df_1], ignore_index=True)
```

```
train_file_paths, test_file_paths, train_labels, test_labels = train_test_split(df_subset["path"], df_subset["label"], test_size=0.2, random_state=42)
```

```
import shutil
```

```
train_dir = "train_data"
if os.path.exists(train_dir):
    shutil.rmtree(train_dir)
os.makedirs(train_dir)
os.makedirs(os.path.join(train_dir, "0"))
os.makedirs(os.path.join(train_dir, "1"))
for file_path, label in zip(train_file_paths, train_labels):
    name = file_path.split("/")[-1]
    if label == "0":
        shutil.copy2(file_path, os.path.join(train_dir, "0", name))
    else:
        shutil.copy2(file_path, os.path.join(train_dir, "1", name))
```

21BAI1007_PCAM_KFolds.ipynb

```
[ ] test_dir = "test_data"
if os.path.exists(test_dir):
    shutil.rmtree(test_dir)
os.makedirs(test_dir)
os.makedirs(os.path.join(test_dir, "0"))
os.makedirs(os.path.join(test_dir, "1"))
for file_path, label in zip(test_file_paths, test_labels):
    name = file_path.split("/")[-1]
    if label == "0":
        shutil.copy2(file_path, os.path.join(test_dir, "0", name))
    else:
        shutil.copy2(file_path, os.path.join(test_dir, "1", name))

[ ] label_0_path_train = '/content/train_data/0'
label_1_path_train = '/content/train_data/1'

[ ] def load_images(file_paths, label):
    images = []
    labels = []
    for file_path in file_paths:
        image = cv2.imread(file_path, cv2.IMREAD_UNCHANGED)
        image = cv2.resize(image, (64, 64))
        images.append(image)
        labels.append(label)
    return np.array(images), np.array(labels)

[ ] label_0_file_paths = [os.path.join(label_0_path_train, file) for file in os.listdir(label_0_path_train)]
X_label_0, Y_label_0 = load_images(label_0_file_paths, 0)

label_1_file_paths = [os.path.join(label_1_path_train, file) for file in os.listdir(label_1_path_train)]
X_label_1, Y_label_1 = load_images(label_1_file_paths, 1)
```

21BAI1007_PCAM_KFolds.ipynb

```
[ ] x = np.concatenate((X_label_0, X_label_1), axis=0)
y = np.concatenate((Y_label_0, Y_label_1), axis=0)

{ } ▾ Creating the model - 21BAI1007

[ ] seed = 7
np.random.seed(seed)

kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=seed)
cvscores = []

[ ] from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
x = scaler.fit_transform(x.reshape(x.shape[0], -1)).reshape(x.shape)
```

21BAI1007_PCAM_KFolds.ipynb

```
[ ] + Code + Text

{ } for train, test in kfold.split(x, y):
    model = Sequential()

    model.add(Conv2D(filters=16, kernel_size=(3,3)))
    model.add(Conv2D(filters=16, kernel_size=(3,3)))
    model.add(MaxPooling2D(pool_size=(2,2)))

    model.add(Conv2D(filters=32, kernel_size=(3,3)))
    model.add(Conv2D(filters=32, kernel_size=(3,3)))
    model.add(Flatten())
    model.add(Dense(1, activation='sigmoid'))

    model.build(input_shape=(32, 64, 64, 3))

    model.compile(loss='binary_crossentropy', metrics=['accuracy'])

    model.fit(x[train], y[train], steps_per_epoch=687, epochs = 5, validation_data = (x[test], y[test]), validation_steps=171, verbose=1)

    scores = model.evaluate(x[test], y[test], verbose = 0)
    print("%%: %.2f%%" % (model.metrics_names[1], scores[1]*100))
    cvscores.append(scores[1] * 100)

    Epoch 1/5
684/687 [=====>...] - ETA: 0s - loss: 0.6426 - accuracy: 0.6825WARNING:tensorflow:Your input ran out of data; interrupting training. Make sure that your dataset or generator can generate at least 687 samples! This is the last epoch.
687/687 [=====] - 12s 8ms/step - loss: 0.6420 - accuracy: 0.6830 - val_loss: 0.5749 - val_accuracy: 0.7184
Epoch 2/5
687/687 [=====] - 3s 5ms/step - loss: 0.5617 - accuracy: 0.7304
Epoch 3/5
687/687 [=====] - 4s 6ms/step - loss: 0.5161 - accuracy: 0.7607
Epoch 4/5
687/687 [=====] - 5s 7ms/step - loss: 0.4701 - accuracy: 0.7890
Epoch 5/5
619/687 [=====>...] - ETA: 0s - loss: 0.4416 - accuracy: 0.8054WARNING:tensorflow:Your input ran out of data; interrupting training. Make sure that your dataset or generator can generate at least 687 samples! This is the last epoch.
687/687 [=====] - 5s 7ms/step - loss: 0.4410 - accuracy: 0.8056
accuracy: 70.81%
```

The screenshot shows a Jupyter notebook titled "21BAI1007_PCAM_KFolds.ipynb". The code cell contains training logs for a neural network across 5 epochs. The logs show increasing accuracy from 71.78% to 70.89% with standard deviation +/- 0.90%. The notebook interface includes tabs for "Code" and "Text", and various toolbar icons.

```
+ Code + Text
[ ] 687/687 [=====] - 3s 5ms/step - loss: 0.4446 - accuracy: 0.8064
[ ] accuracy: 71.78%
Epoch 1/5
686/687 [=====>...] - ETA: 0s - loss: 0.6466 - accuracy: 0.6761WARNING:tensorflow:Your input ran out of data; interrupting training. Make sure that your dataset or generator can generate at least 687 more elements. Stop training
687/687 [=====] - 8s 10ms/step - loss: 0.6465 - accuracy: 0.6760 - val_loss: 0.6033 - val_accuracy: 0.6969
Epoch 2/5
687/687 [=====] - 5s 8ms/step - loss: 0.5627 - accuracy: 0.7285
687/687 [=====] - 5s 8ms/step - loss: 0.5155 - accuracy: 0.7557
Epoch 3/5
687/687 [=====] - 5s 7ms/step - loss: 0.4660 - accuracy: 0.7908
Epoch 4/5
614/687 [=====>...] - ETA: 0s - loss: 0.4429 - accuracy: 0.8044WARNING:tensorflow:Your input ran out of data; interrupting training. Make sure that your dataset or generator can generate at least 173 more elements. Stop training
687/687 [=====] - 4s 6ms/step - loss: 0.4430 - accuracy: 0.8046
accuracy: 71.78%
Epoch 1/5
687/687 [=====] - ETA: 0s - loss: 0.6399 - accuracy: 0.6831WARNING:tensorflow:Your input ran out of data; interrupting training. Make sure that your dataset or generator can generate at least 687 more elements. Stop training
687/687 [=====] - 8s 10ms/step - loss: 0.6399 - accuracy: 0.6831 - val_loss: 0.6012 - val_accuracy: 0.7091
Epoch 2/5
687/687 [=====] - 5s 7ms/step - loss: 0.5611 - accuracy: 0.7296
Epoch 3/5
687/687 [=====] - 3s 5ms/step - loss: 0.5176 - accuracy: 0.7622
Epoch 4/5
687/687 [=====] - 3s 5ms/step - loss: 0.4713 - accuracy: 0.7882
Epoch 5/5
614/687 [=====>...] - ETA: 0s - loss: 0.4388 - accuracy: 0.8843WARNING:tensorflow:Your input ran out of data; interrupting training. Make sure that your dataset or generator can generate at least 173 more elements. Stop training
687/687 [=====] - 3s 5ms/step - loss: 0.4394 - accuracy: 0.8035
accuracy: 70.78%
Epoch 1/5
682/687 [=====>...] - ETA: 0s - loss: 0.6569 - accuracy: 0.6658WARNING:tensorflow:Your input ran out of data; interrupting training. Make sure that your dataset or generator can generate at least 175 more elements. Stop training
687/687 [=====] - 9s 11ms/step - loss: 0.6560 - accuracy: 0.6665 - val_loss: 0.6050 - val_accuracy: 0.7041
Epoch 2/5
687/687 [=====] - 5s 7ms/step - loss: 0.5702 - accuracy: 0.7246
[ ] import numpy as np
print("%2f%% (+/- %.2f%%)" % (np.mean(cvscores), np.std(cvscores)))
70.89% (+/- 0.90%)
```

Results

Successfully performed classification on the above datasets and the corresponding models have been improvised by using K folds and Grid search.