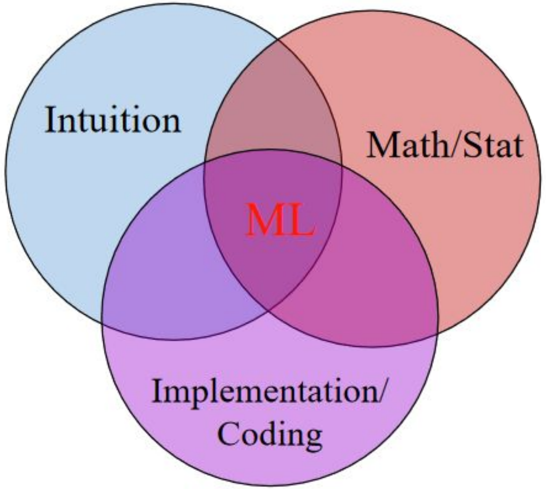
Machine Learning: A Gentle Introduction to Image Classification

Henry Booker Room

May 14th, 2023

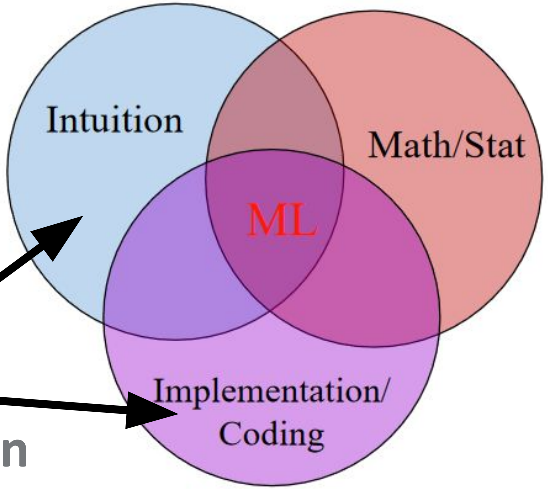
Machine Learning

The capability of a machine to imitate intelligent human behavior



Machine Learning

The capability of a machine to imitate intelligent human behavior

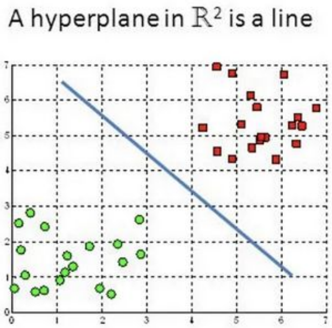


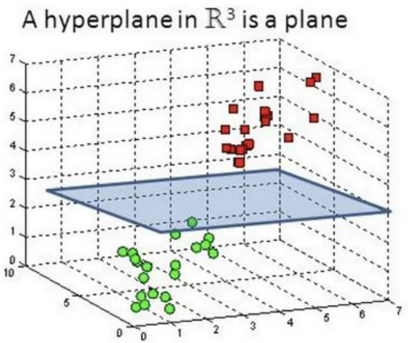
We will be focusing on these two today!

Machine Learning Classification

Intuition

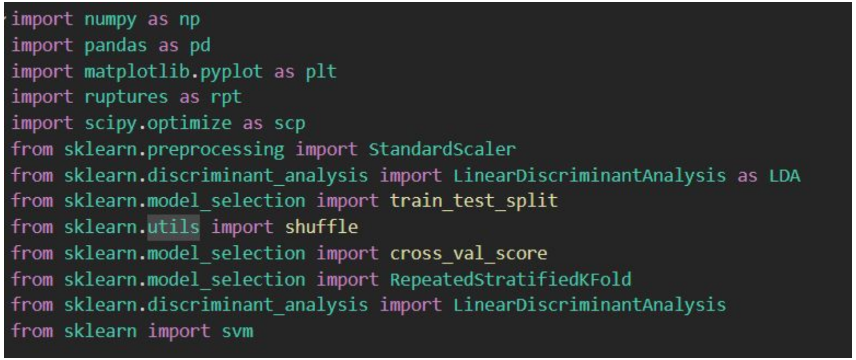
Dividing data by a “decision boundary”





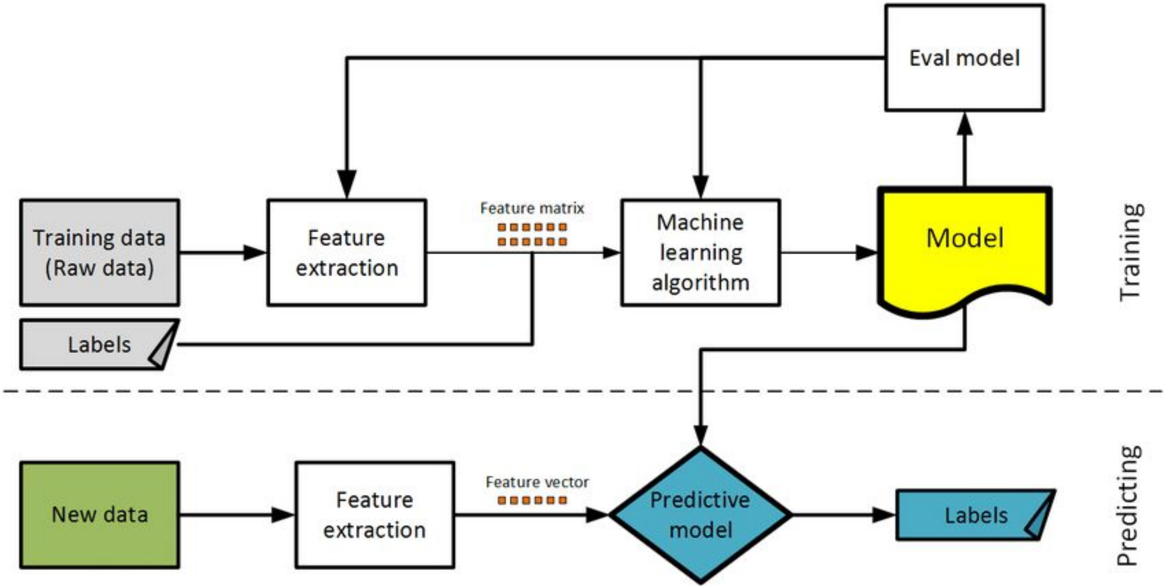
Implementation/Coding

Python packages



```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import ruptures as rpt
import scipy.optimize as scp
from sklearn.preprocessing import StandardScaler
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn import svm
```

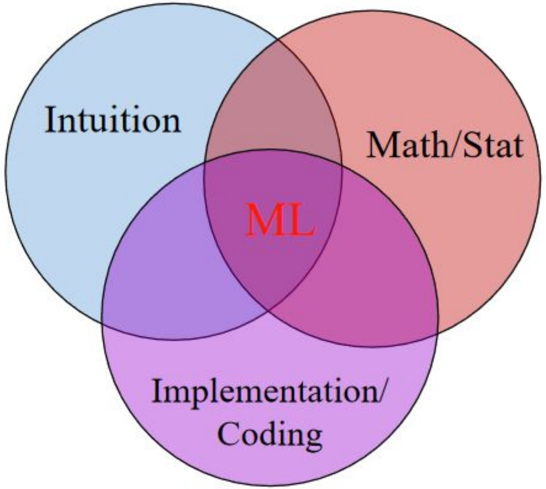
Machine Learning in a Nutshell



1. **Raw data**
2. **Feature Extraction**
3. **Training**
4. **Testing**

Today: Machine Learning in Image recognition

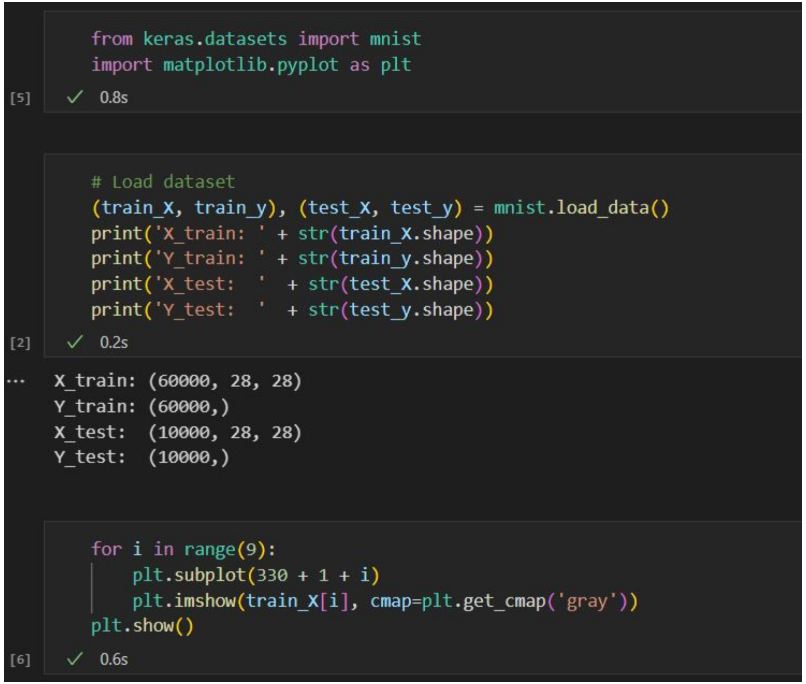
https://github.com/darintsui/Workshops/tree/main/Gentle%20Introduction%20to%20Image%20Classification



If you want to follow along!

1. Raw Data

MNIST: large database of handwritten digits from 0-9



```
from keras.datasets import mnist
import matplotlib.pyplot as plt
```

[5] ✓ 0.8s

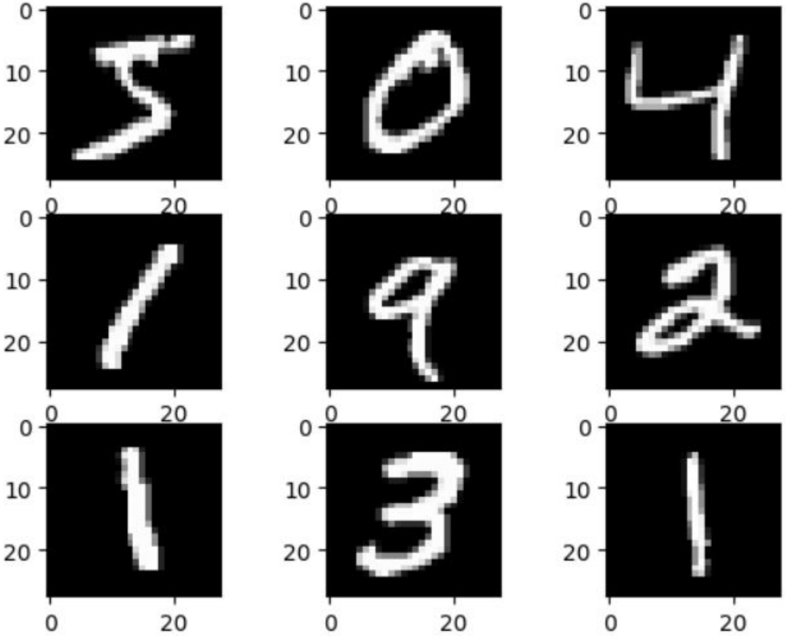
```
# Load dataset
(train_X, train_y), (test_X, test_y) = mnist.load_data()
print('X_train: ' + str(train_X.shape))
print('Y_train: ' + str(train_y.shape))
print('X_test:  '  + str(test_X.shape))
print('Y_test:  '  + str(test_y.shape))
```

[2] ✓ 0.2s

```
X_train: (60000, 28, 28)
Y_train: (60000,)
X_test:  (10000, 28, 28)
Y_test:  (10000,)
```

```
for i in range(9):
    plt.subplot(330 + 1 + i)
    plt.imshow(train_X[i], cmap=plt.get_cmap('gray'))
plt.show()
```

[6] ✓ 0.6s



2. Feature Extraction

Features are the “inputs” you put into your machine learning classifier

- Here, our features are the pixels in our images

Before we continue, we need to talk about how to classify our images: one-hot encoding

2. Feature Extraction - One-Hot Encoding

One-hot encoding: assign an image as either a 0 or 1

- Ex. Imagine a survey: Are you at this workshop?

2. Feature Extraction - One-Hot Encoding

One-hot encoding: assign an image as either a 0 or 1

- Ex. Imagine a survey: Are you at this workshop?

Answers
Yes
Yes
Yes

2. Feature Extraction - One-Hot Encoding

One-hot encoding: assign an image as either a 0 or 1

- Ex. Imagine a survey: Are you at this workshop?

Answers
Yes
Yes
Yes

3 rows
(responses)

Yes	No
1	0
1	0
1	0

2 columns
(total options)

2. Feature Extraction - One-Hot Encoding

One-hot encoding: assign an image as either a 0 or 1

- Ex. What’s your favorite color?

Answers
Red
Green
Blue
Red

2. Feature Extraction - One-Hot Encoding

One-hot encoding: assign an image as either a 0 or 1

- Ex. What’s your favorite color?

Answers
Red
Green
Blue
Red

4 rows (responses)

Red	Green	Blue
1	0	0
0	1	0
0	0	1
1	0	0

3 columns (total options)

2. Feature Extraction - One-Hot Encoding

One-hot encoding: assign an image as either a 0 or 1

- Ex. MNIST data

Answers
0
1
9
2
9

2. Feature Extraction - One-Hot Encoding

One-hot encoding: assign an image as either a 0 or 1

- Ex. MNIST data

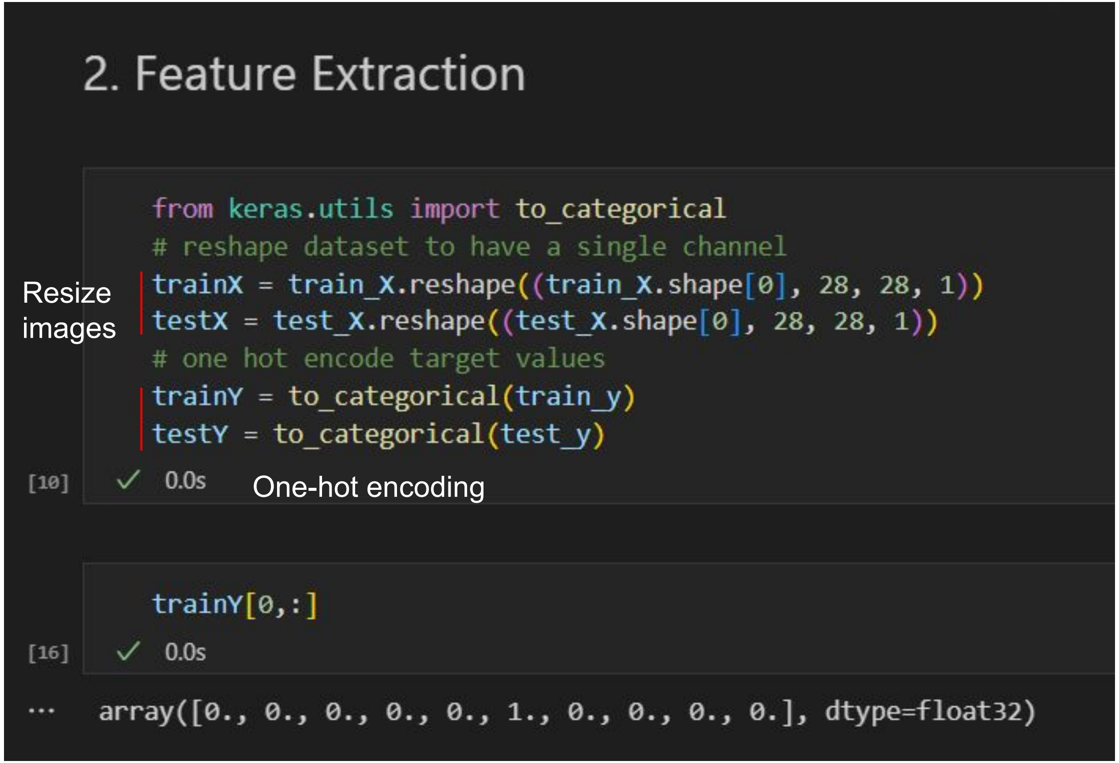
Answers
0
1
9
2
9

5 rows (responses)

0	1	2	3	4	5	6	7	8	9
1	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1

10 columns (total options)

2. Feature Extraction - One-Hot Encoding



2. Feature Extraction

```
from keras.utils import to_categorical
# reshape dataset to have a single channel
trainX = train_X.reshape((train_X.shape[0], 28, 28, 1))
testX = test_X.reshape((test_X.shape[0], 28, 28, 1))
# one hot encode target values
trainY = to_categorical(train_y)
testY = to_categorical(test_y)
```

Resize images

One-hot encoding

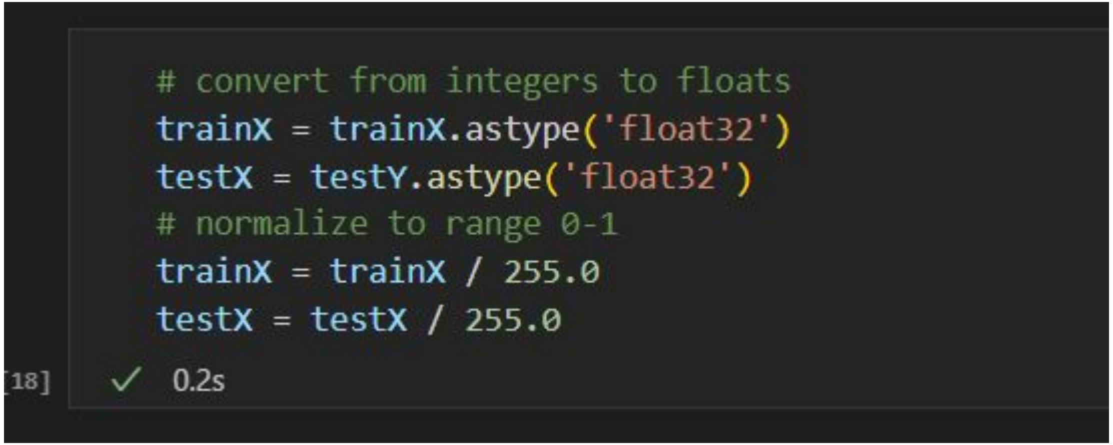
```
trainY[0,:]
```

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

2. Feature Extraction - Normalize

Before we can train our model, we have to normalize our data

- img = img/255



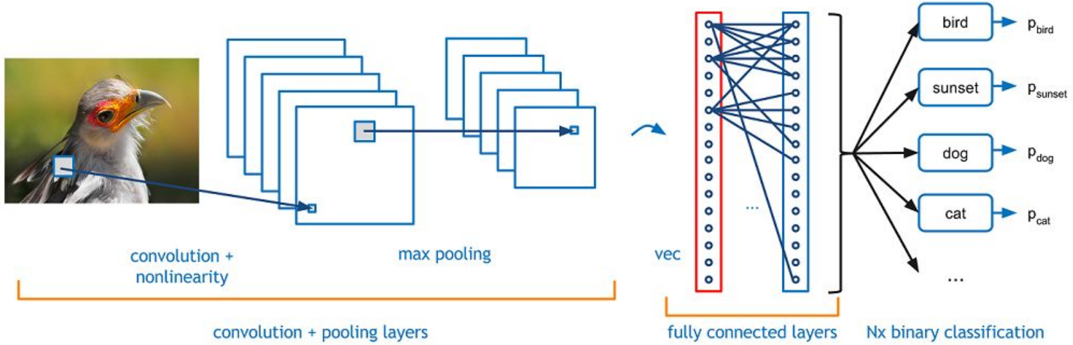
```
# convert from integers to floats
trainX = trainX.astype('float32')
testX = testY.astype('float32')
# normalize to range 0-1
trainX = trainX / 255.0
testX = testX / 255.0
```

[18] ✓ 0.2s

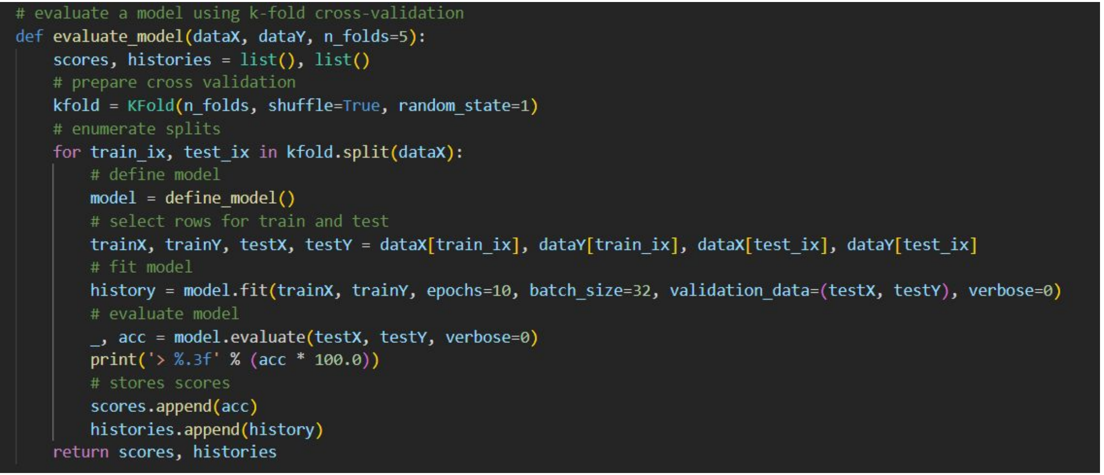
3. Train - Convolutional Neural Networks

Often used in image classification for its ability to generalize images (but can be applied to any input!)

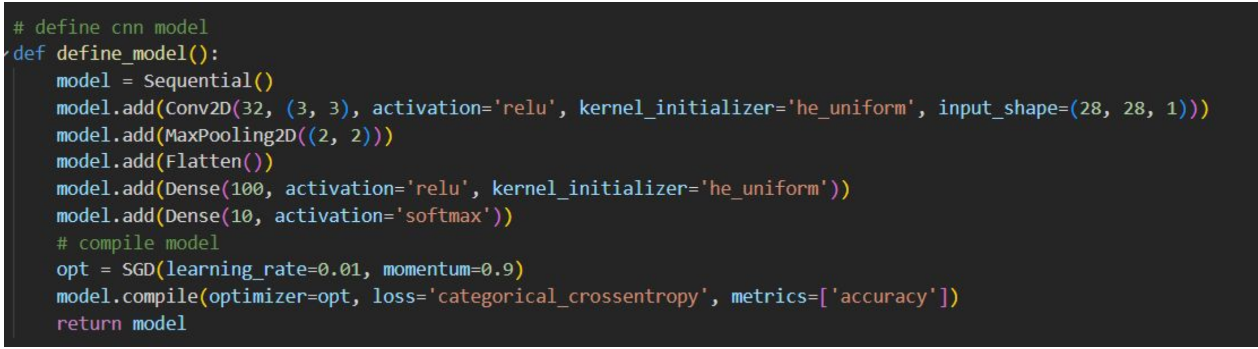
- Come up to me after you want to talk about the math and theory, but for now we’ll just go with a coding implementation



3. Train - Convolutional Neural Networks

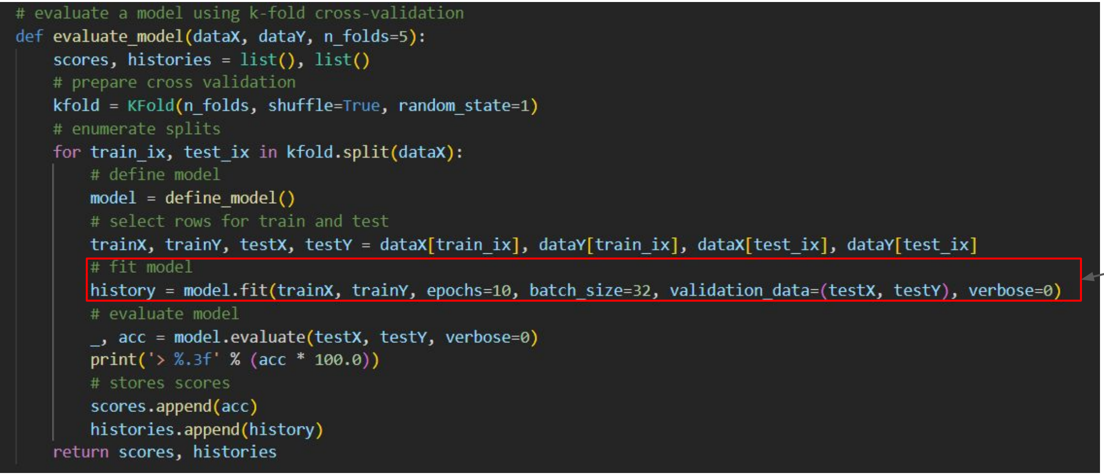


```
# evaluate a model using k-fold cross-validation
def evaluate_model(dataX, dataY, n_folds=5):
    scores, histories = list(), list()
    # prepare cross validation
    kfold = KFold(n_folds, shuffle=True, random_state=1)
    # enumerate splits
    for train_ix, test_ix in kfold.split(dataX):
        # define model
        model = define_model()
        # select rows for train and test
        trainX, trainY, testX, testY = dataX[train_ix], dataY[train_ix], dataX[test_ix], dataY[test_ix]
        # fit model
        history = model.fit(trainX, trainY, epochs=10, batch_size=32, validation_data=(testX, testY), verbose=0)
        # evaluate model
        _, acc = model.evaluate(testX, testY, verbose=0)
        print('> %.3f' % (acc * 100.0))
        # stores scores
        scores.append(acc)
        histories.append(history)
    return scores, histories
```



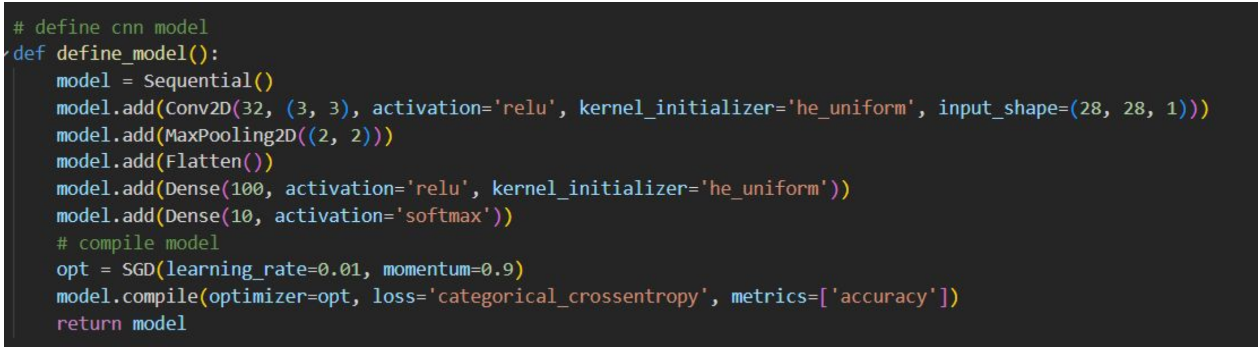
```
# define cnn model
def define_model():
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', input_shape=(28, 28, 1)))
    model.add(MaxPooling2D((2, 2)))
    model.add(Flatten())
    model.add(Dense(100, activation='relu', kernel_initializer='he_uniform'))
    model.add(Dense(10, activation='softmax'))
    # compile model
    opt = SGD(learning_rate=0.01, momentum=0.9)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

3. Train - Convolutional Neural Networks



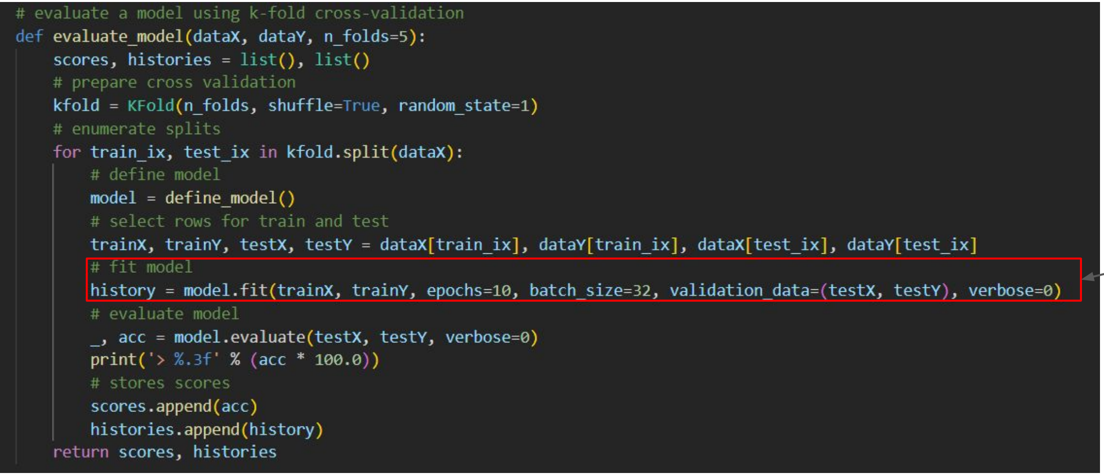
```
# evaluate a model using k-fold cross-validation
def evaluate_model(dataX, dataY, n_folds=5):
    scores, histories = list(), list()
    # prepare cross validation
    kfold = KFold(n_folds, shuffle=True, random_state=1)
    # enumerate splits
    for train_ix, test_ix in kfold.split(dataX):
        # define model
        model = define_model()
        # select rows for train and test
        trainX, trainY, testX, testY = dataX[train_ix], dataY[train_ix], dataX[test_ix], dataY[test_ix]
        # fit model
        history = model.fit(trainX, trainY, epochs=10, batch_size=32, validation_data=(testX, testY), verbose=0)
        # evaluate model
        _, acc = model.evaluate(testX, testY, verbose=0)
        print('> %.3f' % (acc * 100.0))
        # stores scores
        scores.append(acc)
        histories.append(history)
    return scores, histories
```

Fit CNN model with 10 epochs and a batch size of 32



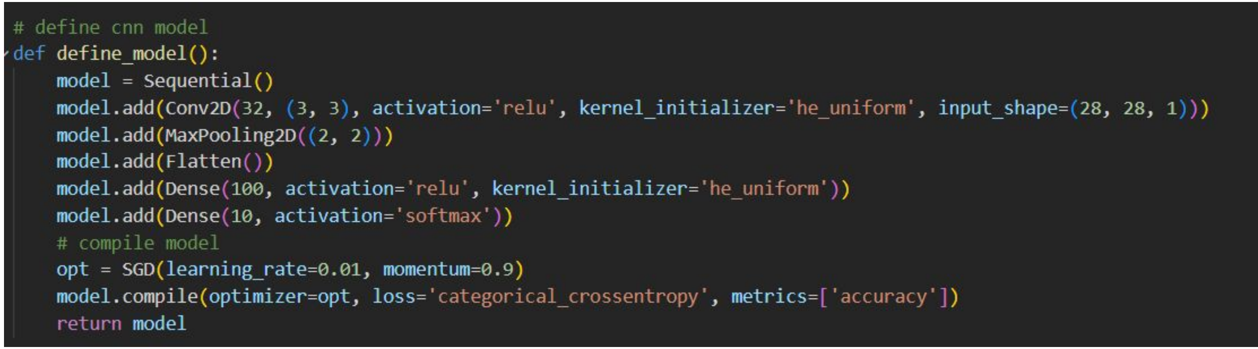
```
# define cnn model
def define_model():
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', input_shape=(28, 28, 1)))
    model.add(MaxPooling2D((2, 2)))
    model.add(Flatten())
    model.add(Dense(100, activation='relu', kernel_initializer='he_uniform'))
    model.add(Dense(10, activation='softmax'))
    # compile model
    opt = SGD(learning_rate=0.01, momentum=0.9)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

3. Train - Convolutional Neural Networks



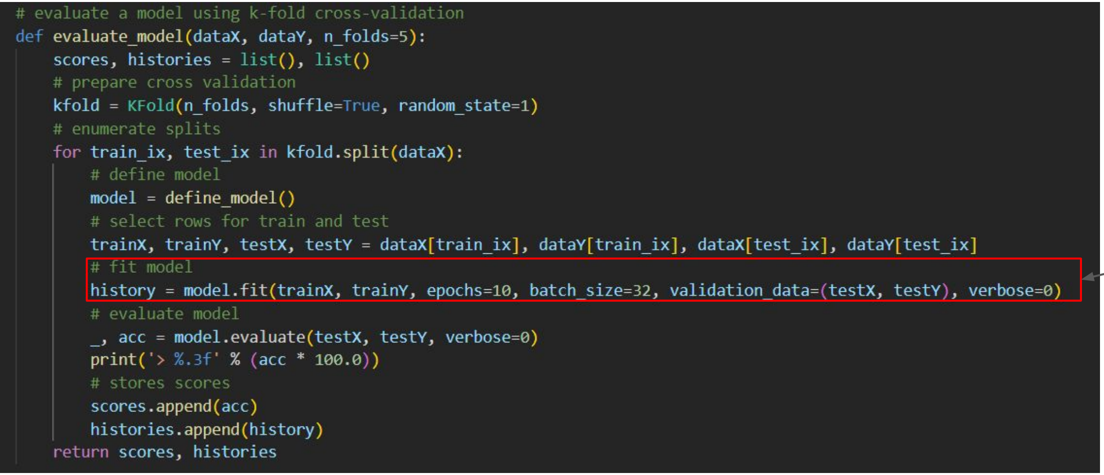
```
# evaluate a model using k-fold cross-validation
def evaluate_model(dataX, dataY, n_folds=5):
    scores, histories = list(), list()
    # prepare cross validation
    kfold = KFold(n_folds, shuffle=True, random_state=1)
    # enumerate splits
    for train_ix, test_ix in kfold.split(dataX):
        # define model
        model = define_model()
        # select rows for train and test
        trainX, trainY, testX, testY = dataX[train_ix], dataY[train_ix], dataX[test_ix], dataY[test_ix]
        # fit model
        history = model.fit(trainX, trainY, epochs=10, batch_size=32, validation_data=(testX, testY), verbose=0)
        # evaluate model
        _, acc = model.evaluate(testX, testY, verbose=0)
        print('> %.3f' % (acc * 100.0))
        # stores scores
        scores.append(acc)
        histories.append(history)
    return scores, histories
```

Fit CNN model with 10 runs and inputting 32 training samples at a time (saves time by inputting as a batch)



```
# define cnn model
def define_model():
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', input_shape=(28, 28, 1)))
    model.add(MaxPooling2D((2, 2)))
    model.add(Flatten())
    model.add(Dense(100, activation='relu', kernel_initializer='he_uniform'))
    model.add(Dense(10, activation='softmax'))
    # compile model
    opt = SGD(learning_rate=0.01, momentum=0.9)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

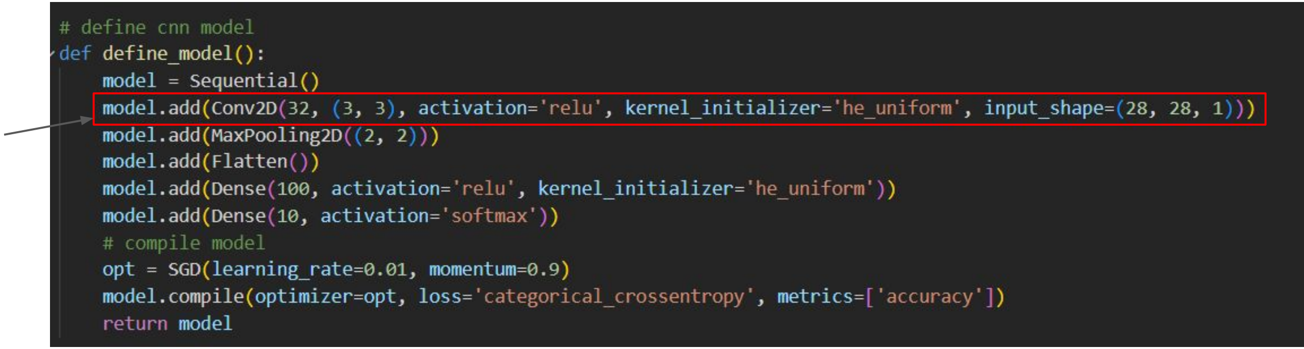
3. Train - Convolutional Neural Networks



```
# evaluate a model using k-fold cross-validation
def evaluate_model(dataX, dataY, n_folds=5):
    scores, histories = list(), list()
    # prepare cross validation
    kfold = KFold(n_folds, shuffle=True, random_state=1)
    # enumerate splits
    for train_ix, test_ix in kfold.split(dataX):
        # define model
        model = define_model()
        # select rows for train and test
        trainX, trainY, testX, testY = dataX[train_ix], dataY[train_ix], dataX[test_ix], dataY[test_ix]
        # fit model
        history = model.fit(trainX, trainY, epochs=10, batch_size=32, validation_data=(testX, testY), verbose=0)
        # evaluate model
        _, acc = model.evaluate(testX, testY, verbose=0)
        print('> %.3f' % (acc * 100.0))
        # stores scores
        scores.append(acc)
        histories.append(history)
    return scores, histories
```

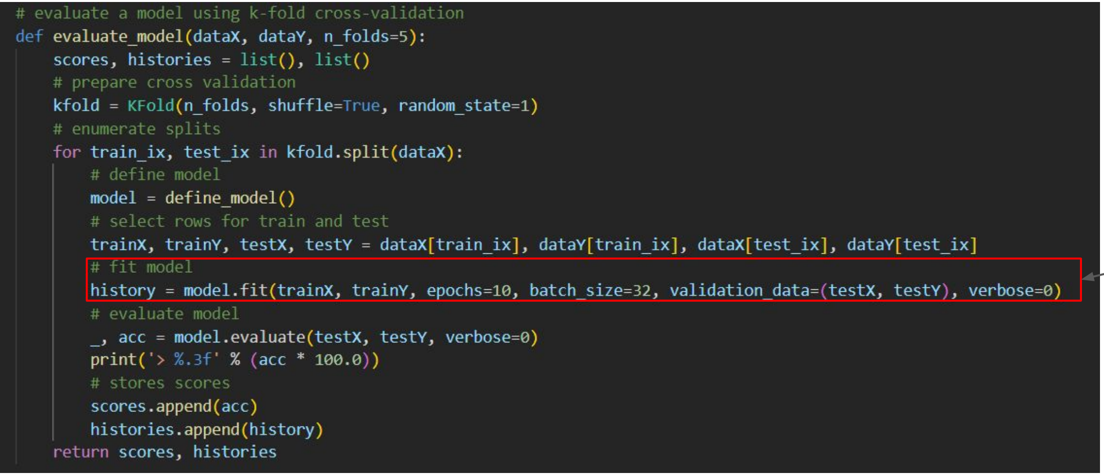
Fit CNN model with 10 runs and inputting 32 training samples at a time (saves time by inputting as a batch)

Call your CNN model with a batch size of 32, using an activation of ReLu



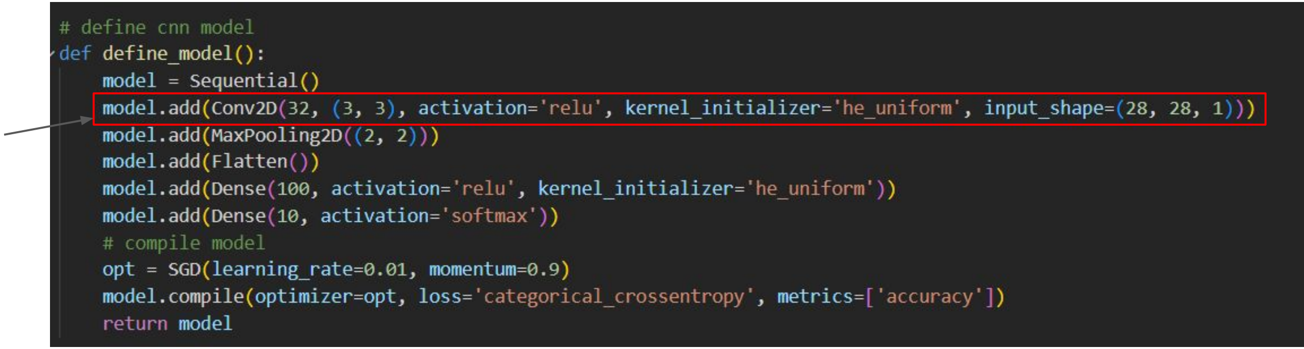
```
# define cnn model
def define_model():
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', input_shape=(28, 28, 1)))
    model.add(MaxPooling2D((2, 2)))
    model.add(Flatten())
    model.add(Dense(100, activation='relu', kernel_initializer='he_uniform'))
    model.add(Dense(10, activation='softmax'))
    # compile model
    opt = SGD(learning_rate=0.01, momentum=0.9)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

3. Train - Convolutional Neural Networks



```
# evaluate a model using k-fold cross-validation
def evaluate_model(dataX, dataY, n_folds=5):
    scores, histories = list(), list()
    # prepare cross validation
    kfold = KFold(n_folds, shuffle=True, random_state=1)
    # enumerate splits
    for train_ix, test_ix in kfold.split(dataX):
        # define model
        model = define_model()
        # select rows for train and test
        trainX, trainY, testX, testY = dataX[train_ix], dataY[train_ix], dataX[test_ix], dataY[test_ix]
        # fit model
        history = model.fit(trainX, trainY, epochs=10, batch_size=32, validation_data=(testX, testY), verbose=0)
        # evaluate model
        _, acc = model.evaluate(testX, testY, verbose=0)
        print('> %.3f' % (acc * 100.0))
        # stores scores
        scores.append(acc)
        histories.append(history)
    return scores, histories
```

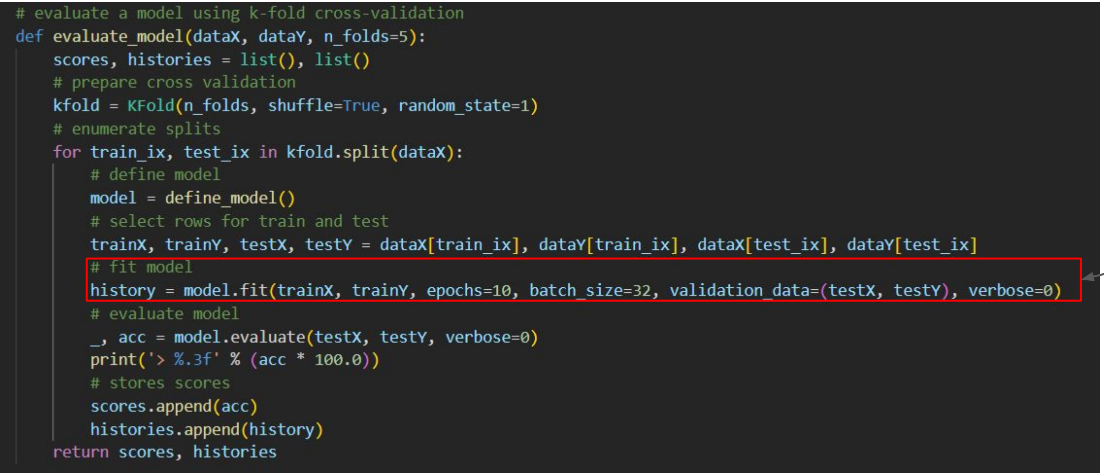
Fit CNN model with 10 runs and inputting 32 training samples at a time (saves time by inputting as a batch)



```
# define cnn model
def define_model():
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', input_shape=(28, 28, 1)))
    model.add(MaxPooling2D((2, 2)))
    model.add(Flatten())
    model.add(Dense(100, activation='relu', kernel_initializer='he_uniform'))
    model.add(Dense(10, activation='softmax'))
    # compile model
    opt = SGD(learning_rate=0.01, momentum=0.9)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

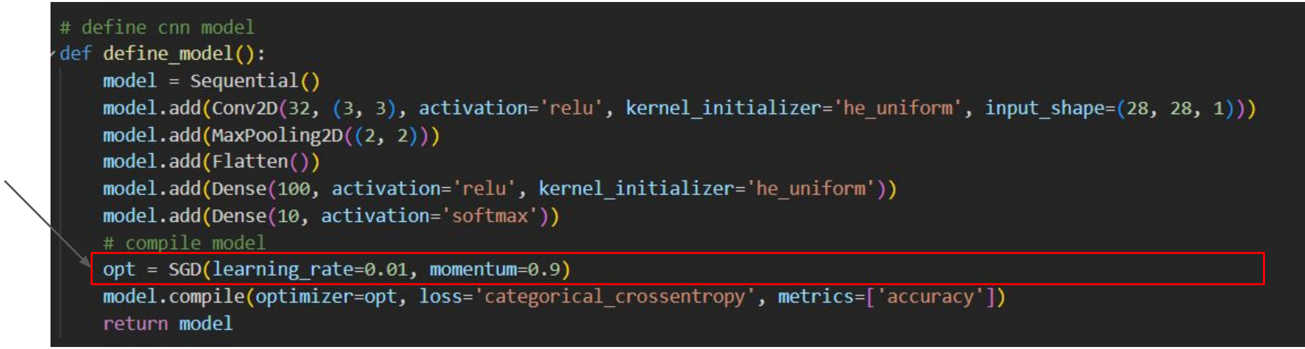
Call your CNN model with a batch size of 32 by mapping your input to another space

3. Train - Convolutional Neural Networks



```
# evaluate a model using k-fold cross-validation
def evaluate_model(dataX, dataY, n_folds=5):
    scores, histories = list(), list()
    # prepare cross validation
    kfold = KFold(n_folds, shuffle=True, random_state=1)
    # enumerate splits
    for train_ix, test_ix in kfold.split(dataX):
        # define model
        model = define_model()
        # select rows for train and test
        trainX, trainY, testX, testY = dataX[train_ix], dataY[train_ix], dataX[test_ix], dataY[test_ix]
        # fit model
        history = model.fit(trainX, trainY, epochs=10, batch_size=32, validation_data=(testX, testY), verbose=0)
        # evaluate model
        _, acc = model.evaluate(testX, testY, verbose=0)
        print('> %.3f' % (acc * 100.0))
        # stores scores
        scores.append(acc)
        histories.append(history)
    return scores, histories
```

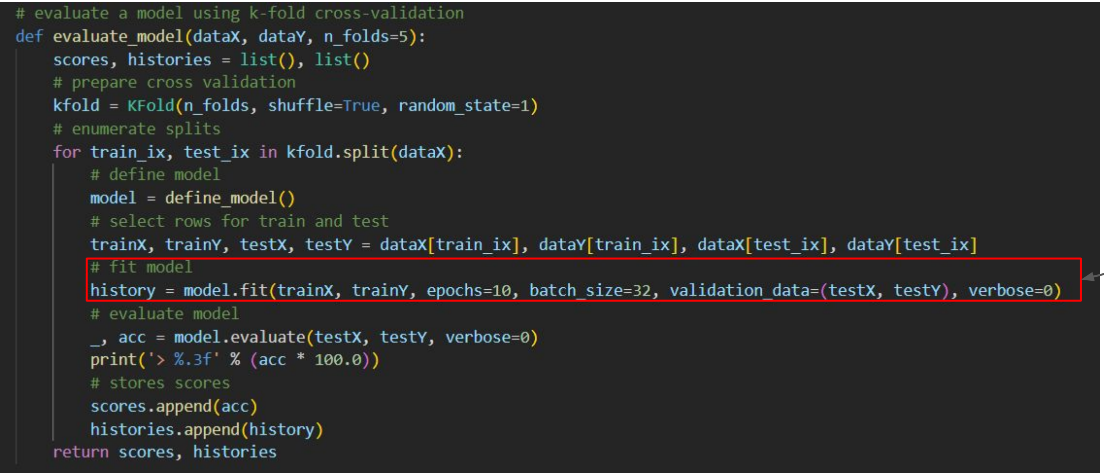
Fit CNN model with 10 runs and inputting 32 training samples at a time (saves time by inputting as a batch)



```
# define cnn model
def define_model():
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', input_shape=(28, 28, 1)))
    model.add(MaxPooling2D((2, 2)))
    model.add(Flatten())
    model.add(Dense(100, activation='relu', kernel_initializer='he_uniform'))
    model.add(Dense(10, activation='softmax'))
    # compile model
    opt = SGD(learning_rate=0.01, momentum=0.9)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

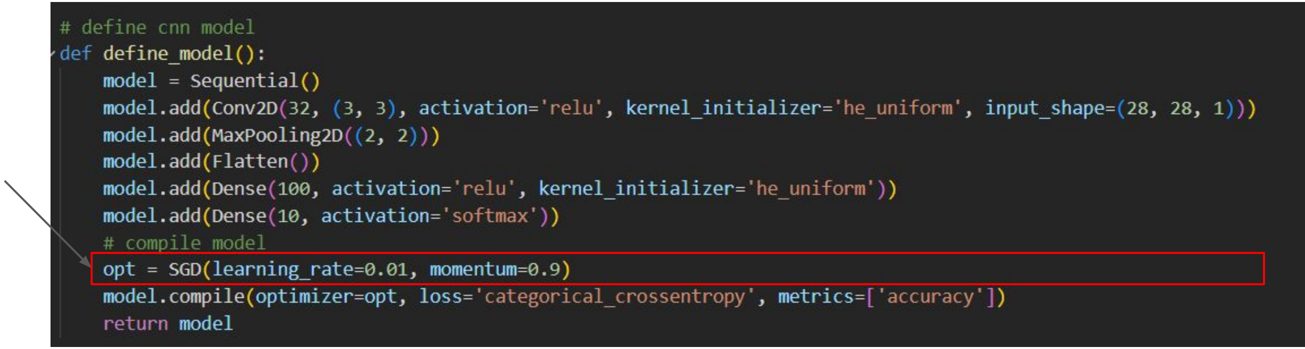
Train your model with a learning rate of 0.01 and using SGD

3. Train - Convolutional Neural Networks



```
# evaluate a model using k-fold cross-validation
def evaluate_model(dataX, dataY, n_folds=5):
    scores, histories = list(), list()
    # prepare cross validation
    kfold = KFold(n_folds, shuffle=True, random_state=1)
    # enumerate splits
    for train_ix, test_ix in kfold.split(dataX):
        # define model
        model = define_model()
        # select rows for train and test
        trainX, trainY, testX, testY = dataX[train_ix], dataY[train_ix], dataX[test_ix], dataY[test_ix]
        # fit model
        history = model.fit(trainX, trainY, epochs=10, batch_size=32, validation_data=(testX, testY), verbose=0)
        # evaluate model
        _, acc = model.evaluate(testX, testY, verbose=0)
        print('> %.3f' % (acc * 100.0))
        # stores scores
        scores.append(acc)
        histories.append(history)
    return scores, histories
```

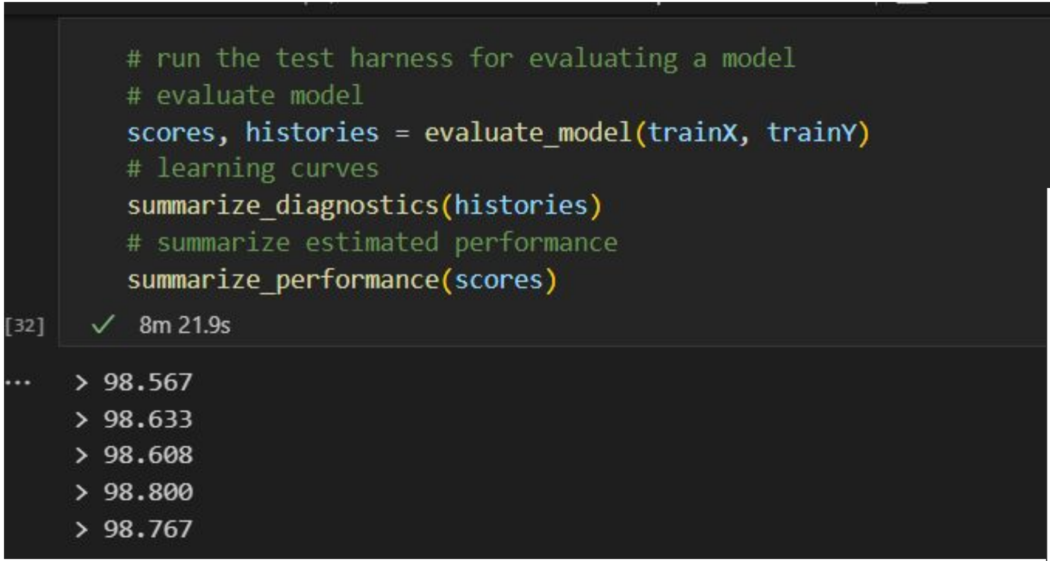
Fit CNN model with 10 runs and inputting 32 training samples at a time (saves time by inputting as a batch)



```
# define cnn model
def define_model():
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', input_shape=(28, 28, 1)))
    model.add(MaxPooling2D((2, 2)))
    model.add(Flatten())
    model.add(Dense(100, activation='relu', kernel_initializer='he_uniform'))
    model.add(Dense(10, activation='softmax'))
    # compile model
    opt = SGD(learning_rate=0.01, momentum=0.9)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

Train your model at a slow rate and converges model to the optimal solution

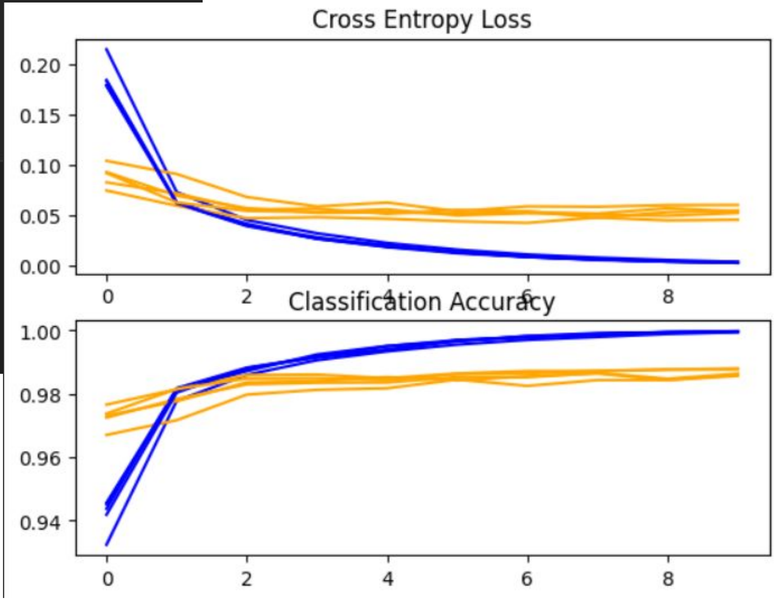
4. Test



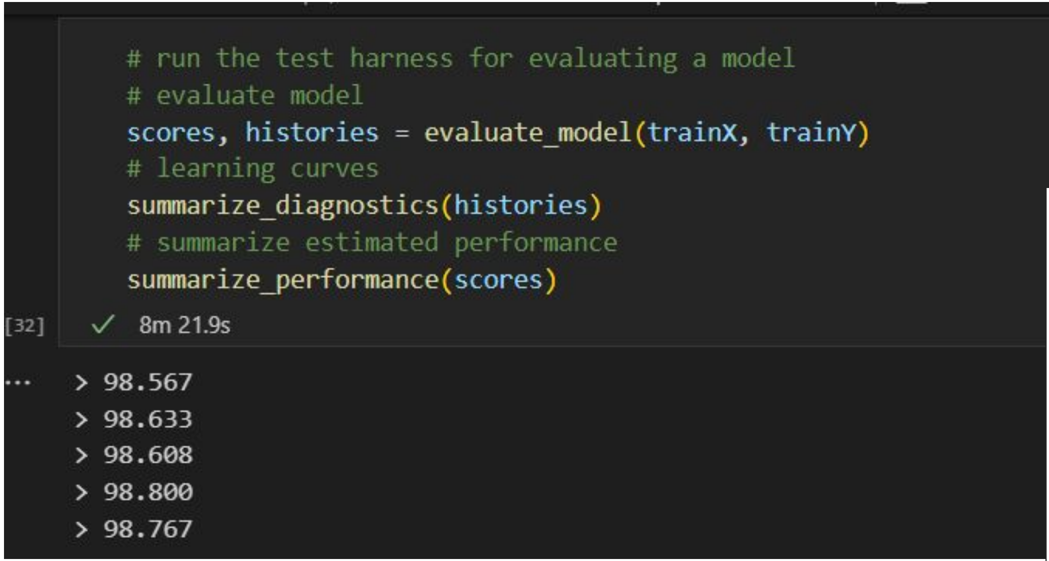
```
# run the test harness for evaluating a model
# evaluate model
scores, histories = evaluate_model(trainX, trainY)
# learning curves
summarize_diagnostics(histories)
# summarize estimated performance
summarize_performance(scores)
```

[32] ✓ 8m 21.9s

```
> 98.567
> 98.633
> 98.608
> 98.800
> 98.767
```



4. Test

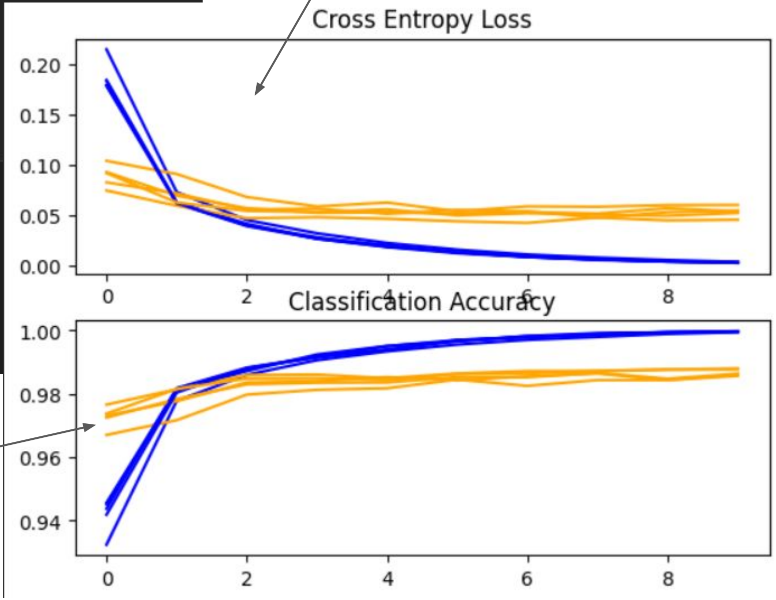


```
# run the test harness for evaluating a model
# evaluate model
scores, histories = evaluate_model(trainX, trainY)
# learning curves
summarize_diagnostics(histories)
# summarize estimated performance
summarize_performance(scores)
```

```
[32]  ✓  8m 21.9s

...  > 98.567
     > 98.633
     > 98.608
     > 98.800
     > 98.767
```

Generally, want this to go down



Generally, want this to go up

Concluding Thoughts

Wow, that was cool but most of that floated over my head. What can I do?

Concluding Thoughts

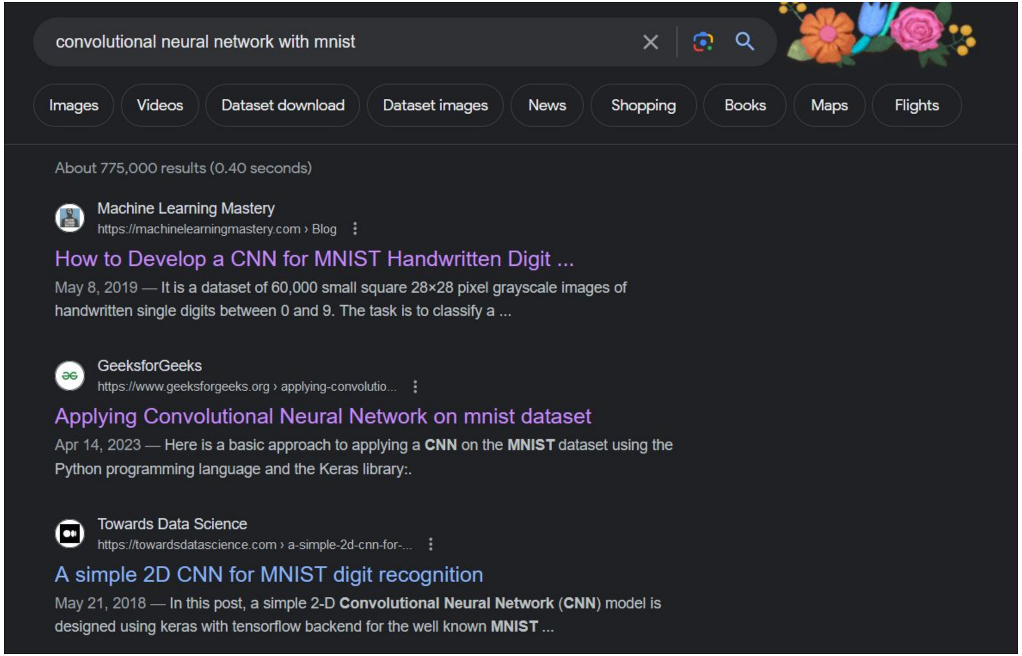
Wow, that was cool but most of that floated over my head. What can I do?

- Online resources

Concluding Thoughts

Wow, that was cool but most of that floated over my head. What can I do?

- Online resources



About 775,000 results (0.40 seconds)

Machine Learning Mastery
https://machinelearningmastery.com › Blog

How to Develop a CNN for MNIST Handwritten Digit ...

May 8, 2019 — It is a dataset of 60,000 small square 28×28 pixel grayscale images of handwritten single digits between 0 and 9. The task is to classify a ...

GeeksforGeeks
https://www.geeksforgeeks.org › applying-convolutio...

Applying Convolutional Neural Network on mnist dataset

Apr 14, 2023 — Here is a basic approach to applying a **CNN** on the **MNIST** dataset using the Python programming language and the Keras library:.

Towards Data Science
https://towardsdatascience.com › a-simple-2d-cnn-for-...

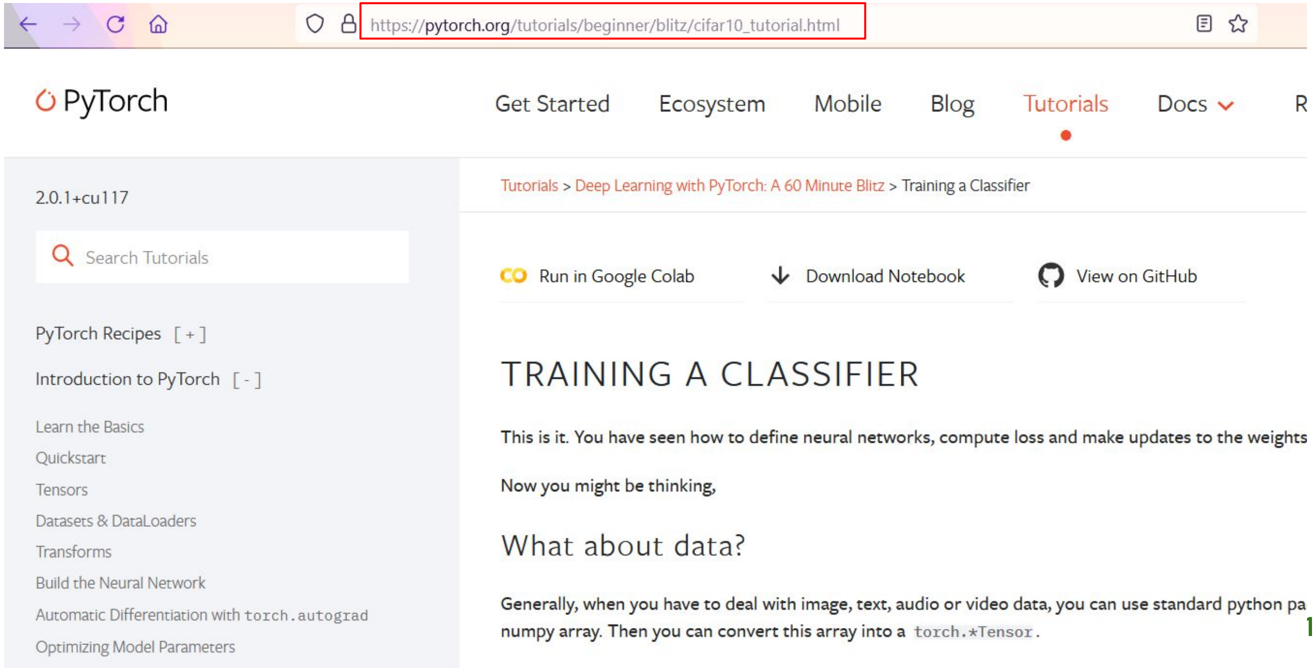
A simple 2D CNN for MNIST digit recognition

May 21, 2018 — In this post, a simple 2-D **Convolutional Neural Network** (**CNN**) model is designed using keras with tensorflow backend for the well known **MNIST** ...

Concluding Thoughts

Wow, that was cool but most of that floated over my head. What can I do?

- Online resources



https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html

TRAINING A CLASSIFIER

This is it. You have seen how to define neural networks, compute loss and make updates to the weights

Now you might be thinking,

What about data?

Generally, when you have to deal with image, text, audio or video data, you can use standard python pa
numpy array. Then you can convert this array into a `torch.*Tensor`.

Concluding Thoughts

Wow, that was cool but most of that floated over my head. What can I do?

- Online resources
- Play around with the code!! Definitely not the most optimal way to do CNN
    - Suggestions to Google to optimize the code (and for your own projects!!)
        - Reduce batch size
        - Decrease learning rate (make sure you have a powerful PC first, otherwise increase it)
        - Switch SGD out for Adam
        - Switch ReLu out for Leaky ReLu

Concluding Thoughts

Wow, that was cool but most of that floated over my head. What can I do?

- Online resources
- Play around with the code!! Definitely not the most optimal way to do CNN
    - Suggestions to Google to optimize the code (and for your own projects!!)
        - Reduce batch size
        - Decrease learning rate (make sure you have a powerful PC first, otherwise increase it)
        - Switch SGD out for Adam
        - Switch ReLu out for Leaky ReLu
- Ask me questions!! I'd love to help :))

Thank you!

dtsui@ucsd.edu