How to Develop a CNN for MNIST Handwritten Digit Classification

by **Jason Brownlee** on May 8, 2019 in **Deep Learning for Computer Vision**

Last Updated on August 24, 2020

How to Develop a Convolutional Neural Network From Scratch for MNIST Handwritten Digit Classification.

The MNIST handwritten digit classification problem is a standard dataset used in computer vision and deep learning.

Although the dataset is effectively solved, it can be used as the basis for learning and practicing how to develop, evaluate, and use convolutional deep learning neural networks for image classification from scratch. This includes how to develop a robust test harness for estimating the performance of the model, how to explore improvements to the model, and how to save the model and later load it to make predictions on new data.

In this tutorial, you will discover how to develop a convolutional neural network for handwritten digit classification from scratch.

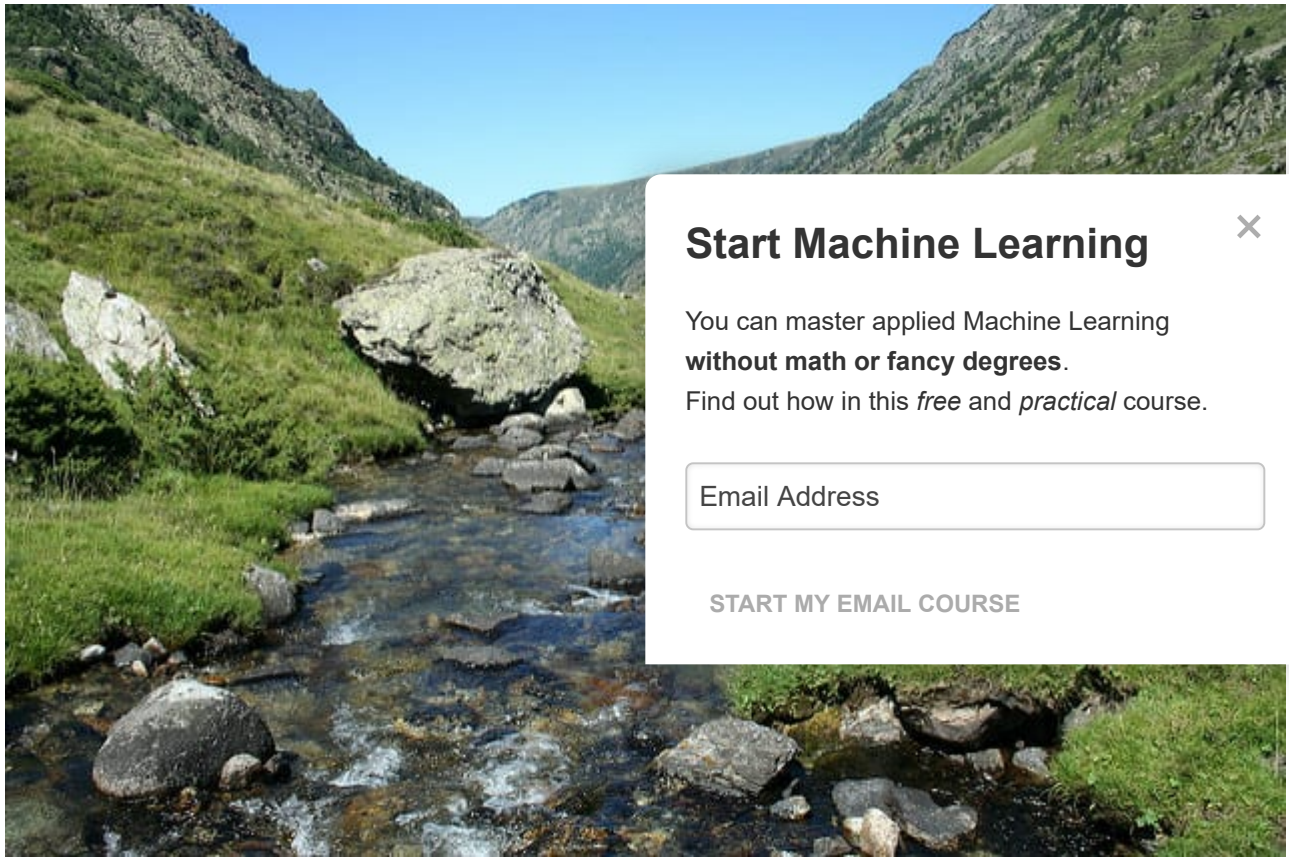
After completing this tutorial, you will know:

- How to develop a test harness to develop a robust evaluation of a model and establish a baseline of performance for a classification task.
- How to explore extensions to a baseline model to improve learning and model capacity.
- How to develop a finalized model, evaluate the performance of the final model, and use it to make predictions on new images.

Kick-start your project with my new book Deep Learning for Computer Vision, including *step-by-step tutorials* and the *Python source code* files for all examples.

Let's get started.

- **Updated Dec/2019**: Updated examples for TensorFlow 2.0 and Keras 2.3.
- **Updated Jan/2020**: Fixed a bug where models were defined outside the cross-validation loop.



How to Develop a Convolutional Neural Network From Scratch for MNIST Handwritten Digit Classification
Photo by Richard Allaway, some rights reserved.

Tutorial Overview

This tutorial is divided into five parts; they are:

1. MNIST Handwritten Digit Classification Dataset
2. Model Evaluation Methodology
3. How to Develop a Baseline Model
4. How to Develop an Improved Model
5. How to Finalize the Model and Make Predictions

Development Environment

This tutorial assumes that you are using standalone Keras running on top of TensorFlow with Python 3. If you need help setting up your development environment see this tutorial:

- How to Setup Your Python Environment for Machine Learning with Anaconda

MNIST Handwritten Digit Classi

The MNIST dataset is an acronym that stands for the Technology dataset.

It is a dataset of 60,000 small square 28×28 pixel gray and 9.

The task is to classify a given image of a handwritten values from 0 to 9, inclusively.

It is a widely used and deeply understood dataset and, for the most part, is "*solved*." Top-performing models are deep learning convolutional neural networks that achieve a classification accuracy of above 99%, with an error rate between 0.4 %and 0.2% on the hold out test dataset.

The example below loads the MNIST dataset using the Keras API and creates a plot of the first nine images in the training dataset.

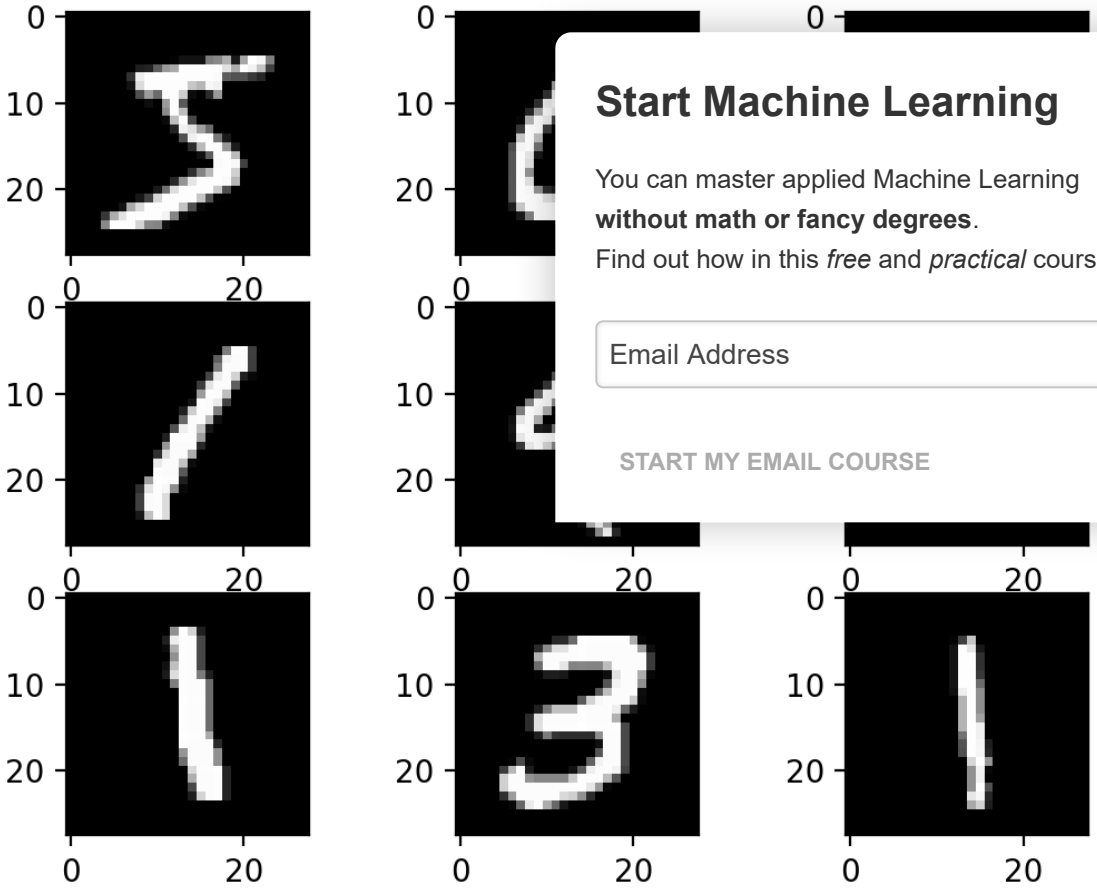
```
# example of loading the mnist dataset
from keras.datasets import mnist
from matplotlib import pyplot
# load dataset
(trainX, trainy), (testX, testy) = mnist.load_data()
# summarize loaded dataset
print('Train: X=%s, y=%s' % (trainX.shape, trainy.shape))
print('Test: X=%s, y=%s' % (testX.shape, testy.shape))
# plot first few images
for i in range(9):
    # define subplot
    pyplot.subplot(330 + 1 + i)
    # plot raw pixel data
    pyplot.imshow(trainX[i], cmap=pyplot.get_cmap('gray'))
# show the figure
pyplot.show()
```

Running the example loads the MNIST train and test d

We can see that there are 60,000 examples in the training dataset and 10,000 in the test dataset and that images are indeed square with 28×28 pixels.

```
Train: X=(60000, 28, 28), y=(60000,)Test: X=(10000, 28, 28), y=(10000,)
```

A plot of the first nine images in the dataset is also created showing the natural handwritten nature of the images to be classified.



Plot of a Subset of Images From the MNIST Dataset

Model Evaluation Methodology

Although the MNIST dataset is effectively solved, it can be a useful starting point for developing and practicing a methodology for solving image classification tasks using convolutional neural networks.

Instead of reviewing the literature on well-performing models on the dataset, we can develop a new model from scratch.

The dataset already has a well-defined train and test dataset that we can use.

In order to estimate the performance of a model for a given training run, we can further split the training set into a train and validation dataset. Performance on the train and validation dataset over each run can then be plotted to provide learning curves and insight into how well a model is learning the problem.

The Keras API supports this by specifying the "*validation_data*" argument to the *model.fit()* function when training the model, that will, in turn, return an object that describes model performance for the chosen loss and metrics on each training epoch.

```
# record model performance on a validation dataset during training
history = model.fit(..., validation_data=(valX, valY))
```

In order to estimate the performance of a model on the problem in general, we can use k-fold cross-validation, perhaps five-fold cross-validation. This will
th
respect to differences in the training and test datasets
ing
algorithm. The performance of a model can be taken a
e
standard deviation, that could be used to estimate a c

We can use the KFold class from the scikit-learn API t
of
a given neural network model. There are many ways t
approach where the *KFold* class is only used to specif


```
# example of k-fold cv for a neural net
data = ...
# prepare cross validation
kfold = KFold(5, shuffle=True, random_state=1)
# enumerate splits
for train_ix, test_ix in kfold.split(data):
        model = ...
    ...
```

We will hold back the actual test dataset and use it as an evaluation of our final model.

How to Develop a Baseline Model

The first step is to develop a baseline model.

This is critical as it both involves developing the infrastructure for the test harness so that any model we design can be evaluated on the dataset, and it establishes a baseline in model performance on the problem, by which all improvements can be compared.

The design of the test harness is modular, and we can develop a separate function for each piece. This allows a given aspect of the test harness to be modified or inter-changed, if we desire, separately from the rest.

We can develop this test harness with five key elements. They are the loading of the dataset, the preparation of the dataset, the definition of the model, the evaluation of the model, and the presentation of results.

Load Dataset

We know some things about the dataset.

For example, we know that the images are all pre-aligned (e.g. each image only contains a hand-drawn digit), that the images all have the same square size of 28×28 pixels, and that the images are grayscale.

Therefore, we can load the images and reshape the data arrays to have a single color channel.

```
# load dataset
(trainX, trainY), (testX, testY) = mnist.load_data()
# reshape dataset to have a single channel
trainX = trainX.reshape((trainX.shape[0], 28, 28, 1))
testX = testX.reshape((testX.shape[0], 28, 28,
```

We also know that there are 10 classes and that class

We can, therefore, use a one hot encoding for the clas er into a 10 element binary vector with a 1 for the index c s. We can achieve this with the *to_categorical()* utility fur

```
# one hot encode target values
trainY = to_categorical(trainY)
testY = to_categorical(testY)
```

The *load_dataset()* function implements these behaviors and can be used to load the dataset.

```
# load train and test dataset
def load_dataset():
    # load dataset
    (trainX, trainY), (testX, testY) = mnist.load_data()
    # reshape dataset to have a single channel
    trainX = trainX.reshape((trainX.shape[0], 28, 28, 1))
    testX = testX.reshape((testX.shape[0], 28, 28, 1))
    # one hot encode target values
    trainY = to_categorical(trainY)
    testY = to_categorical(testY)
    return trainX, trainY, testX, testY
```

Prepare Pixel Data

We know that the pixel values for each image in the dataset are unsigned integers in the range between black and white, or 0 and 255.

We do not know the best way to scale the pixel values for modeling, but we know that some scaling will be required.

A good starting point is to normalize the pixel values of grayscale images, e.g. rescale them to the range [0,1]. This involves first converting the data type from unsigned integers to floats, then dividing the pixel values by the maximum value.

```
# convert from integers to floats
train_norm = train.astype('float32')
test_norm = test.astype('float32')
# normalize to range 0-1
train_norm = train_norm / 255.0
test_norm = test_norm / 255.0
```

The *prep_pixels()* function below implements these behaviors and is provided with the pixel values for both the train and test datasets that will need to be scaled.

```
# scale pixels
def prep_pixels(train, test):
    # convert from integers to floats
    train_norm = train.astype('float32')
    test_norm = test.astype('float32')
    # normalize to range 0-1
    train_norm = train_norm / 255.0
    test_norm = test_norm / 255.0
    # return normalized images
    return train_norm, test_norm
```

This function must be called to prepare the pixel value

Define Model

Next, we need to define a baseline convolutional neur

The model has two main aspects: the feature extraction front end comprised of convolutional and pooling layers, and the classifier backend that will make a prediction.

For the convolutional front-end, we can start with a single convolutional layer with a small filter size (3,3) and a modest number of filters (32) followed by a max pooling layer. The filter maps can then be flattened to provide features to the classifier.

Given that the problem is a multi-class classification task, we know that we will require an output layer with 10 nodes in order to predict the probability distribution of an image belonging to each of the 10 classes. This will also require the use of a softmax activation function. Between the feature extractor and the output layer, we can add a dense layer to interpret the features, in this case with 100 nodes.

All layers will use the ReLU activation function and the He weight initialization scheme, both best practices.

We will use a conservative configuration for the stochastic gradient descent optimizer with a learning rate of 0.01 and a momentum of 0.9. The categorical cross-entropy loss function will be optimized, suitable for multi-class classification, and we will monitor the classification accuracy metric, which is appropriate given we have the same number of examples in each of the 10 classes.

The *define_model()* function below will define and return this model.

```
# define cnn model
```

```
def define_model():
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', input_shap
    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(lr=0.01, momentum=0.9)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

Evaluate Model

After the model is defined, we need to evaluate it.

The model will be evaluated using five-fold cross-valic set
baseline for both repeated evaluation and to not be sc
will be 20% of the training dataset, or about 12,000 ex this
problem.

The training dataset is shuffled prior to being split, anc hat
any model we evaluate will have the same train and te
apples comparison between models.

We will train the baseline model for a modest 10 traini s.
The test set for each fold will be used to evaluate the so
that we can later create learning curves, and at the end of the run, so that we can estimate the performance of the model. As such, we will keep track of the resulting history from each run, as well as the classification accuracy of the fold.

The *evaluate_model()* function below implements these behaviors, taking the training dataset as arguments and returning a list of accuracy scores and training histories that can be later summarized.

```
# 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[
        # fit model
        history = model.fit(trainX, trainY, epochs=10, batch_size=32, validation_data=(testX, t
        # 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
```

Present Results

Once the model has been evaluated, we can present the results.

There are two key aspects to present: the diagnostics of the learning behavior of the model during training and the estimation of the model performance. These can be implemented using separate functions.

First, the diagnostics involve creating a line plot showing model performance on the train and test set during each fold of the k-fold cross-validation. These plots are valuable for getting an idea of whether a model is overfitting, underfitting, or has a good fit for the dataset.

We will create a single figure with two subplots, one fo… model performance on the training dataset and orange… dataset. The *summarize_diagnostics()* function below… training histories.


```
# plot diagnostic learning curves
def summarize_diagnostics(histories):
    for i in range(len(histories)):
        # plot loss
        pyplot.subplot(2, 1, 1)
        pyplot.title('Cross Entropy Loss')
        pyplot.plot(histories[i].history['los…
        pyplot.plot(histories[i].history['val…
        # plot accuracy
        pyplot.subplot(2, 1, 2)
        pyplot.title('Classification Accuracy')
        pyplot.plot(histories[i].history['accuracy'], color='blue', label='train')
        pyplot.plot(histories[i].history['val_accuracy'], color='orange', label='test')
    pyplot.show()
```

Next, the classification accuracy scores collected during each fold can be summarized by calculating the mean and standard deviation. This provides an estimate of the average expected performance of the model trained on this dataset, with an estimate of the average variance in the mean. We will also summarize the distribution of scores by creating and showing a box and whisker plot.

The *summarize_performance()* function below implements this for a given list of scores collected during model evaluation.

```
# summarize model performance
def summarize_performance(scores):
    # print summary
    print('Accuracy: mean=%.3f std=%.3f, n=%d' % (mean(scores)*100, std(scores)*100, len(scores)
    # box and whisker plots of results
    pyplot.boxplot(scores)
    pyplot.show()
```

Complete Example

We need a function that will drive the test harness.

This involves calling all of the define functions.

```
# run the test harness for evaluating a model
def run_test_harness():
    # load dataset
    trainX, trainY, testX, testY = load_dataset()
    # prepare pixel data
    trainX, testX = prep_pixels(trainX, testX)
    # evaluate model
    scores, histories = evaluate_model(trainX, trainY)
    # learning curves
    summarize_diagnostics(histories)
    # summarize estimated performance
    summarize_performance(scores)
```

We now have everything we need; the complete code ork model on the MNIST dataset is listed below.

```
# baseline cnn model for mnist
from numpy import mean
from numpy import std
from matplotlib import pyplot
from sklearn.model_selection import KFold
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.optimizers import SGD

# load train and test dataset
def load_dataset():
    # load dataset
    (trainX, trainY), (testX, testY) = mnist.load_data()
    # reshape dataset to have a single channel
    trainX = trainX.reshape((trainX.shape[0], 28, 28, 1))
    testX = testX.reshape((testX.shape[0], 28, 28, 1))
    # one hot encode target values
    trainY = to_categorical(trainY)
    testY = to_categorical(testY)
    return trainX, trainY, testX, testY

# scale pixels
def prep_pixels(train, test):
    # convert from integers to floats
    train_norm = train.astype('float32')
    test_norm = test.astype('float32')
    # normalize to range 0-1
    train_norm = train_norm / 255.0
    test_norm = test_norm / 255.0
    # return normalized images
    return train_norm, test_norm

# define cnn model
def define_model():
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='
    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(lr=0.01, momentum=0.9)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
    return model

# 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
        # fit model
        history = model.fit(trainX, trainY,
        # evaluate model
        _, acc = model.evaluate(testX, testY
        print('> %.3f' % (acc * 100.0))
        # stores scores
        scores.append(acc)
        histories.append(history)
    return scores, histories

# plot diagnostic learning curves
def summarize_diagnostics(histories):
    for i in range(len(histories)):
        # plot loss
        pyplot.subplot(2, 1, 1)
        pyplot.title('Cross Entropy Loss')
        pyplot.plot(histories[i].history['loss'], color='blue', label='train')
        pyplot.plot(histories[i].history['val_loss'], color='orange', label='test')
        # plot accuracy
        pyplot.subplot(2, 1, 2)
        pyplot.title('Classification Accuracy')
        pyplot.plot(histories[i].history['accuracy'], color='blue', label='train')
        pyplot.plot(histories[i].history['val_accuracy'], color='orange', label='test')
    pyplot.show()

# summarize model performance
def summarize_performance(scores):
    # print summary
    print('Accuracy: mean=%.3f std=%.3f, n=%d' % (mean(scores)*100, std(scores)*100, len(score
    # box and whisker plots of results
    pyplot.boxplot(scores)
    pyplot.show()

# run the test harness for evaluating a model
def run_test_harness():
    # load dataset
    trainX, trainY, testX, testY = load_dataset()
    # prepare pixel data
    trainX, testX = prep_pixels(trainX, testX)
    # evaluate model
    scores, histories = evaluate_model(trainX, trainY)
    # learning curves
    summarize_diagnostics(histories)
    # summarize estimated performance
    summarize_performance(scores)

```


Start Machine Learning

```
# entry point, run the test harness
run_test_harness()
```

Running the example prints the classification accuracy for each fold of the cross-validation process. This is helpful to get an idea that the model evaluation is progressing.

Note: Your results may vary given the stochastic nature of the algorithm or evaluation procedure, or differences in numerical precision. Consider running the example a few times and compare the average outcome.

We can see two cases where the model achieves perfect skill and one case where it achieved lower than 98% accuracy. These are good results.

```
> 98.550
> 98.600
> 98.642
> 98.850
> 98.742
```

Next, a diagnostic plot is shown, giving insight into the

In this case, we can see that the model generally achi
converging. There is no obvious sign of over- or under

Start Machine Learning

Loss and Accuracy Learning Curves for the Baseline Model During k-Fold Cross-Validation

Next, a summary of the model performance is calculated.

We can see in this case, the model has an estimated skill of about 98.6%, which is reasonable.

```
Accuracy: mean=98.677 std=0.107, n=5
```

Finally, a box and whisker plot is created to summarize the distribution of accuracy scores.

Box and Whisker Plot of Accuracy Scores for the Baseline Model Evaluated Using k-Fold Cross-Validation

We now have a robust test harness and a well-performing baseline model.

How to Develop an Improved Model

There are many ways that we might explore improvements to the baseline model.

We will look at areas of model configuration that often result in an improvement, so-called low-hanging fruit. The first is a change to the learning algorithm, and the second is an increase in the depth of the model.

Improvement to Learning

There are many aspects of the learning algorithm that can be explored for improvement.

Perhaps the point of biggest leverage is the learning rate, such as evaluating the impact that smaller or larger values of the learning rate may have, as well as schedules that change the learning rate during training.

Another approach that can rapidly accelerate the learning of a model and can result in large performance improvements is batch normalization. We will evaluate the effect that batch normalization has on our baseline model.

Batch normalization can be used after convolutional and fully connected layers. It has the effect of changing the distribution of the output of the layer, specifically by standardizing the outputs. This has the effect of stabilizing and accelerating the learning process.

We can update the model definition to use batch normalization after the activation function for the convolutional and dense layers of our baseline model. The updated version of *define_model()* function with batch normalization is listed below.

```
# define cnn model
def define_model():
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='
    model.add(BatchNormalization())
    model.add(MaxPooling2D((2, 2)))
    model.add(Flatten())
    model.add(Dense(100, activation='relu', k
    model.add(BatchNormalization())
    model.add(Dense(10, activation='softmax')
    # compile model
    opt = SGD(lr=0.01, momentum=0.9)
    model.compile(optimizer=opt, loss='catego
    return model
```


The complete code listing with this change is provided below.

```
# cnn model with batch normalization for mnist
from numpy import mean
from numpy import std
from matplotlib import pyplot
from sklearn.model_selection import KFold
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.optimizers import SGD
from keras.layers import BatchNormalization

# load train and test dataset
def load_dataset():
    # load dataset
    (trainX, trainY), (testX, testY) = mnist.load_data()
    # reshape dataset to have a single channel
    trainX = trainX.reshape((trainX.shape[0], 28, 28, 1))
    testX = testX.reshape((testX.shape[0], 28, 28, 1))
    # one hot encode target values
    trainY = to_categorical(trainY)
    testY = to_categorical(testY)
    return trainX, trainY, testX, testY

# 
```

```
# scale pixels
def prep_pixels(train, test):
    # convert from integers to floats
    train_norm = train.astype('float32')
    test_norm = test.astype('float32')
    # normalize to range 0-1
    train_norm = train_norm / 255.0
    test_norm = test_norm / 255.0
    # return normalized images
    return train_norm, test_norm

# define cnn model
def define_model():
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', input_sha
    model.add(BatchNormalization())
    model.add(MaxPooling2D((2, 2)))
    model.add(Flatten())
    model.add(Dense(100, activation='relu',
    model.add(BatchNormalization())
    model.add(Dense(10, activation='softmax'
    # compile model
    opt = SGD(lr=0.01, momentum=0.9)
    model.compile(optimizer=opt, loss='categ
    return model

# evaluate a model using k-fold cross-valida
def evaluate_model(dataX, dataY, n_folds=5):
    scores, histories = list(), list()
    # prepare cross validation
    kfold = KFold(n_folds, shuffle=True, ran
    # enumerate splits
    for train_ix, test_ix in kfold.split(dat
        # 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
        # fit model
        history = model.fit(trainX, trainY, epochs=10, batch_size=32, validation_data=(testX,
        # 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

# plot diagnostic learning curves
def summarize_diagnostics(histories):
    for i in range(len(histories)):
        # plot loss
        pyplot.subplot(2, 1, 1)
        pyplot.title('Cross Entropy Loss')
        pyplot.plot(histories[i].history['loss'], color='blue', label='train')
        pyplot.plot(histories[i].history['val_loss'], color='orange', label='test')
        # plot accuracy
        pyplot.subplot(2, 1, 2)
        pyplot.title('Classification Accuracy')
        pyplot.plot(histories[i].history['accuracy'], color='blue', label='train')
        pyplot.plot(histories[i].history['val_accuracy'], color='orange', label='test')
    pyplot.show()

# summarize model performance
def summarize_performance(scores):
    # print summary
```

```
    print('Accuracy: mean=%.3f std=%.3f, n=%d' % (mean(scores)*100, std(scores)*100, len(score
    # box and whisker plots of results
    pyplot.boxplot(scores)
    pyplot.show()

# run the test harness for evaluating a model
def run_test_harness():
    # load dataset
    trainX, trainY, testX, testY = load_dataset()
    # prepare pixel data
    trainX, testX = prep_pixels(trainX, testX)
    # evaluate model
    scores, histories = evaluate_model(trainX, trainY)
    # learning curves
    summarize_diagnostics(histories)
    # summarize estimated performance
    summarize_performance(scores)

# entry point, run the test harness
run_test_harness()
```

Running the example again reports model performanc

Note: Your results may vary given the stochastic natu differences in numerical precision. Consider running th outcome.

We can see perhaps a small drop in model performan validation folds.


```
> 98.475
> 98.608
> 98.683
> 98.783
> 98.667
```

A plot of the learning curves is created, in this case showing that the speed of learning (improvement over epochs) does not appear to be different from the baseline model.

The plots suggest that batch normalization, at least as implemented in this case, does not offer any benefit.

Start Machine Learning

Loss and Accuracy Learning Curves for the BatchNormalization Model During k-Fold Cross-Validation

Next, the estimated performance of the model is presented, showing performance with a slight decrease in the mean accuracy of the model: 98.643 as compared to 98.677 with the baseline model.

```
Accuracy: mean=98.643 std=0.101, n=5
```

Box and Whisker Plot of Accuracy Scores for the BatchNormalization Model Evaluated Using k-Fold Cross-Validation

Increase in Model Depth

There are many ways to change the model configuration in order to explore improvements over the baseline model.

Two common approaches involve changing the capacity of the feature extraction part of the model or changing the capacity or function of the classifier part of the model. Perhaps the point of biggest influence is a change to the feature extractor.

We can increase the depth of the feature extractor part of the model, following a VGG-like pattern of adding more convolutional and pooling layers with the same sized filter, while increasing the number of filters. In this case, we will add a double convolutional layer with 64 filters each, followed by another max pooling layer.

The updated version of the *define_model()* function with this change is listed below.

```
# define cnn model
def define_model():
```

```
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', input_shap
    model.add(MaxPooling2D((2, 2)))
    model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform'))
    model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform'))
    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(lr=0.01, momentum=0.9)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

For completeness, the entire code listing, including this change, is provided below.

```
# deeper cnn model for mnist
from numpy import mean
from numpy import std
from matplotlib import pyplot
from sklearn.model_selection import KFold
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.optimizers import SGD

# load train and test dataset
def load_dataset():
    # load dataset
    (trainX, trainY), (testX, testY) = mnist.load_data()
    # reshape dataset to have a single channel
    trainX = trainX.reshape((trainX.shape[0], 28, 28, 1))
    testX = testX.reshape((testX.shape[0], 28, 28, 1))
    # one hot encode target values
    trainY = to_categorical(trainY)
    testY = to_categorical(testY)
    return trainX, trainY, testX, testY

# scale pixels
def prep_pixels(train, test):
    # convert from integers to floats
    train_norm = train.astype('float32')
    test_norm = test.astype('float32')
    # normalize to range 0-1
    train_norm = train_norm / 255.0
    test_norm = test_norm / 255.0
    # return normalized images
    return train_norm, test_norm

# define cnn model
def define_model():
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', input_sha
    model.add(MaxPooling2D((2, 2)))
    model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform'))
    model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Flatten())
    model.add(Dense(100, activation='relu',
```

```
    model.add(Dense(10, activation='softmax'))
    # compile model
    opt = SGD(lr=0.01, momentum=0.9)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
    return model

# 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
        # fit model
        history = model.fit(trainX, trainY,
        # evaluate model
        _, acc = model.evaluate(testX, testY
        print('> %.3f' % (acc * 100.0))
        # stores scores
        scores.append(acc)
        histories.append(history)
    return scores, histories

# plot diagnostic learning curves
def summarize_diagnostics(histories):
    for i in range(len(histories)):
        # plot loss
        pyplot.subplot(2, 1, 1)
        pyplot.title('Cross Entropy Loss')
        pyplot.plot(histories[i].history['loss'], color='blue', label='train')
        pyplot.plot(histories[i].history['val_loss'], color='orange', label='test')
        # plot accuracy
        pyplot.subplot(2, 1, 2)
        pyplot.title('Classification Accuracy')
        pyplot.plot(histories[i].history['accuracy'], color='blue', label='train')
        pyplot.plot(histories[i].history['val_accuracy'], color='orange', label='test')
    pyplot.show()

# summarize model performance
def summarize_performance(scores):
    # print summary
    print('Accuracy: mean=%.3f std=%.3f, n=%d' % (mean(scores)*100, std(scores)*100, len(scores
    # box and whisker plots of results
    pyplot.boxplot(scores)
    pyplot.show()

# run the test harness for evaluating a model
def run_test_harness():
    # load dataset
    trainX, trainY, testX, testY = load_dataset()
    # prepare pixel data
    trainX, testX = prep_pixels(trainX, testX)
    # evaluate model
    scores, histories = evaluate_model(trainX, trainY)
    # learning curves
    summarize_diagnostics(histories)
    # summarize estimated performance
    summarize_performance(scores)

# entry point, run the test harness
run_test_harness()
```

Running the example reports model performance for each fold of the cross-validation process.

Note: Your results may vary given the stochastic nature of the algorithm or evaluation procedure, or differences in numerical precision. Consider running the example a few times and compare the average outcome.

The per-fold scores may suggest some improvement over the baseline.

```
> 99.058
> 99.042
> 98.883
> 99.192
> 99.133
```

A plot of the learning curves is created, in this case sh
problem, with no clear signs of overfitting. The plots m
helpful.

Loss and Accuracy Learning Curves for the Deeper Model During k-Fold Cross-Validation

Next, the estimated performance of the model is presented, showing a small improvement in performance as compared to the baseline from 98.677 to 99.062, with a small drop in the standard deviation as well.

```
Accuracy: mean=99.062 std=0.104, n=5
```


Box and Whisker Plot of Accuracy Scores for the Deeper Model Evaluated Using k-Fold Cross-Validation

How to Finalize the Model and Make Predictions

The process of model improvement may continue for as long as we have ideas and the time and resources to test them out.

At some point, a final model configuration must be chosen and adopted. In this case, we will choose the deeper model as our final model.

First, we will finalize our model, but fitting a model on the entire training dataset and saving the model to file for later use. We will then load the model and evaluate its performance on the hold out test dataset to get an idea of how well the chosen model actually performs in practice. Finally, we will use the saved model to make a prediction on a single image.

Save Final Model

A final model is typically fit on all available data, such as the combination of all train and test dataset.

In this tutorial, we are intentionally holding back a test dataset so that we can estimate the performance of the final model, which can be a good idea in practice. As such, we will fit our model on the training dataset only.

```
# fit model
model.fit(trainX, trainY, epochs=10, batch_size=32, verbose=0)
```

Once fit, we can save the final model to an H5 file by calling the *save()* function on the model and pass in the chosen filename.

```
# save model
model.save('final_model.h5')
```

Note, saving and loading a Keras model requires that

The complete example of fitting the final deep model c
below.

```
# save the final model to file
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.optimizers import SGD

# load train and test dataset
def load_dataset():
    # load dataset
    (trainX, trainY), (testX, testY) = mnist.load_data()
    # reshape dataset to have a single channel
    trainX = trainX.reshape((trainX.shape[0], 28, 28, 1))
    testX = testX.reshape((testX.shape[0], 28, 28, 1))
    # one hot encode target values
    trainY = to_categorical(trainY)
    testY = to_categorical(testY)
    return trainX, trainY, testX, testY

# scale pixels
def prep_pixels(train, test):
    # convert from integers to floats
    train_norm = train.astype('float32')
    test_norm = test.astype('float32')
    # normalize to range 0-1
    train_norm = train_norm / 255.0
    test_norm = test_norm / 255.0
    # return normalized images
    return train_norm, test_norm

```

```
# define cnn model
def define_model():
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', input_shap
    model.add(MaxPooling2D((2, 2)))
    model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform'))
    model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform'))
    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(lr=0.01, momentum=0.9)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
    return model

# run the test harness for evaluating a model
def run_test_harness():
    # load dataset
    trainX, trainY, testX, testY = load_datas
    # prepare pixel data
    trainX, testX = prep_pixels(trainX, testX
    # define model
    model = define_model()
    # fit model
    model.fit(trainX, trainY, epochs=10, batc
    # save model
    model.save('final_model.h5')

# entry point, run the test harness
run_test_harness()
```


After running this example, you will now have a 1.2-m
current working directory.

Evaluate Final Model

We can now load the final model and evaluate it on the hold out test dataset.

This is something we might do if we were interested in presenting the performance of the chosen model to project stakeholders.

The model can be loaded via the *load_model()* function.

The complete example of loading the saved model and evaluating it on the test dataset is listed below.

```
# evaluate the deep model on the test dataset
from keras.datasets import mnist
from keras.models import load_model
from keras.utils import to_categorical

# load train and test dataset
def load_dataset():
    # load dataset
    (trainX, trainY), (testX, testY) = mnist.load_data()
    # reshape dataset to have a single channel
    trainX = trainX.reshape((trainX.shape[0],
    testX = testX.reshape((testX.shape[0], 28
    # one hot encode target values
```

```
    trainY = to_categorical(trainY)
    testY = to_categorical(testY)
    return trainX, trainY, testX, testY

# scale pixels
def prep_pixels(train, test):
    # convert from integers to floats
    train_norm = train.astype('float32')
    test_norm = test.astype('float32')
    # normalize to range 0-1
    train_norm = train_norm / 255.0
    test_norm = test_norm / 255.0
    # return normalized images
    return train_norm, test_norm

# run the test harness for evaluating a model
def run_test_harness():
    # load dataset
    trainX, trainY, testX, testY = load_datas
    # prepare pixel data
    trainX, testX = prep_pixels(trainX, testX
    # load model
    model = load_model('final_model.h5')
    # evaluate model on test dataset
    _, acc = model.evaluate(testX, testY, ver
    print('> %.3f' % (acc * 100.0))

# entry point, run the test harness
run_test_harness()
```


Running the example loads the saved model and eval

Note: Your results may vary given the stochastic nature of the algorithm or evaluation procedure, or differences in numerical precision. Consider running the example a few times and compare the average outcome.

The classification accuracy for the model on the test dataset is calculated and printed. In this case, we can see that the model achieved an accuracy of 99.090%, or just less than 1%, which is not bad at all and reasonably close to the estimated 99.753% with a standard deviation of about half a percent (e.g. 99% of scores).

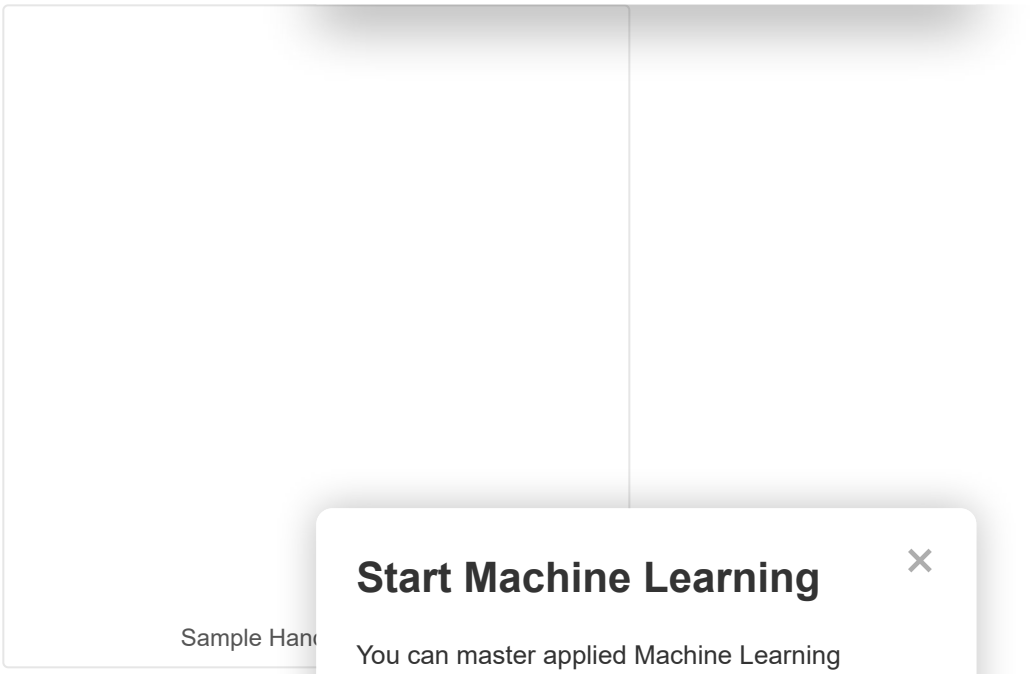
```
> 99.090
```

Make Prediction

We can use our saved model to make a prediction on new images.

The model assumes that new images are grayscale, that they have been aligned so that one image contains one centered handwritten digit, and that the size of the image is square with the size 28×28 pixels.

Below is an image extracted from the MNIST test dataset. You can save it in your current working directory with the filename '*sample_image.png*'.



Sample Hand

- Download the sample image (sample_image.png

We will pretend this is an entirely new and unseen ima e might use our saved model to predict the integer that t

First, we can load the image, force it to be in grayscal e loaded image can then be resized to have a single cha The *load_image()* function implements this and will return the loaded image ready for classification.

Importantly, the pixel values are prepared in the same way as the pixel values were prepared for the training dataset when fitting the final model, in this case, normalized.

```
# load and prepare the image
def load_image(filename):
    # load the image
    img = load_img(filename, grayscale=True, target_size=(28, 28))
    # convert to array
    img = img_to_array(img)
    # reshape into a single sample with 1 channel
    img = img.reshape(1, 28, 28, 1)
    # prepare pixel data
    img = img.astype('float32')
    img = img / 255.0
    return img
```

Next, we can load the model as in the previous section and call the *predict_classes()* function to predict the digit that the image represents.

```
# predict the class
digit = model.predict_classes(img)
```

The complete example is listed below.

```
# make a prediction for a new image.
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.models import load_model

# load and prepare the image
def load_image(filename):
    # load the image
    img = load_img(filename, grayscale=True, target_size=(28, 28))
    # convert to array
    img = img_to_array(img)
    # reshape into a single sample with 1 channel
    img = img.reshape(1, 28, 28, 1)
    # prepare pixel data
    img = img.astype('float32')
    img = img / 255.0
    return img

# load an image and predict the class
def run_example():
    # load the image
    img = load_image('sample_image.png')
    # load model
    model = load_model('final_model.h5')
    # predict the class
    digit = model.predict_classes(img)
    print(digit[0])

# entry point, run the example
run_example()
```


Running the example first loads and prepares the ima at the loaded image represents the digit '7'.

```
7
```

Extensions

This section lists some ideas for extending the tutorial that you may wish to explore.

- **Tune Pixel Scaling**. Explore how alternate pixel scaling methods impact model performance as compared to the baseline model, including centering and standardization.
- **Tune the Learning Rate**. Explore how different learning rates impact the model performance as compared to the baseline model, such as 0.001 and 0.0001.
- **Tune Model Depth**. Explore how adding more layers to the model impact the model performance as compared to the baseline model, such as another block of convolutional and pooling layers or another dense layer in the classifier part of the model.

If you explore any of these extensions, I'd love to know.
Post your findings in the comments below.

Further Reading

This section provides more resources on the topic if y

APIs

- Keras Datasets API
- Keras Datasets Code
- sklearn.model_selection.KFold API

Articles

- MNIST database, Wikipedia.
- Classification datasets results, What is the class of this image?

Summary

In this tutorial, you discovered how to develop a convo
classification from scratch.

Specifically, you learned:

- How to develop a test harness to develop a robus
performance for a classification task.
- How to explore extensions to a baseline model to
- How to develop a finalized model, evaluate the pe
predictions on new images.

Do you have any questions?
Ask your questions in the comments below and I will do my best to answer.

About Jason Brownlee

Jason Brownlee, PhD is a machine learning specialist who teaches developers how to get results with modern machine learning methods via hands-on tutorials.

View all posts by Jason Brownlee →

‹ How to Visualize Filters and Feature Maps in Convolutional Neural Networks

tion ›

140 Responses to *How to Develop a CN… Classification*

SHAHEEN ALHIRMIZY May 10, 2019 at 4:32 pm

Dear Jason Thank You Very Much For your w… for Edge detection from scratch using real images.

REPLY

Jason Brownlee May 11, 2019 at 6:05 am #

Great suggestion, thanks.

REPLY

SHAHEEN ALHIRMIZY May 10, 2019 at 4:47 pm #

I have two questions :

first suppose I have images sizes equal to 7611 x 7811 how to deal with this big size images in CNN models.

second question about if our images different sizes not same size.

REPLY

Jason Brownlee May 11, 2019 at 6:06 am #

I recommend reducing the size of your images first, before modeling, e.g. less than 1000 pixels, even less than 500 pixels if you can.

I then recommend normalizing images to the same size.

REPLY

vinay September 30, 2019 at 11:47 am #

to reduce the size of the images, which approach is better. Using machine learning or normal image compression methods.

REPLY

Jason Brownlee September 30, 2019 at 2:26 pm #

Use image compression algorithm

SHAHEEN ALHIRMIZY May 11, 2019 at 4:13 pm

when we reducing the size of images that me

Jason Brownlee May 12, 2019 at 6:38 am #

Often it does.

REPLY

Taranpreetkaur October 12, 2019 at 10:57 pm #

How u make this project please help us … We can't understand how we make this project … Taranpreetkaur

REPLY

Jason Brownlee October 13, 2019 at 8:30 am #

What problem are you having exactly?

REPLY

Ajoy October 20, 2019 at 4:08 am #

Original error was: No module named _multiarray_umath, keras .imageprocessing Importing the numpy c-extensions failed.

Any ideas please

Jason Brownlee October 20, 2019 at 6:23 am #

REPLY

Sorry to hear that.

Perhaps try checking that your version of Keras and TensorFlow are up to date?

Ajoy October 20, 2019 at 7:18 pm #

REPLY

Sorry dont worry, I somehow got to resolve it. I think I reinstalled stuff.
I have another problem though. How do I add data for learning if in case the data predicted the wrong value
pelase. I think I need the advise for both how to add as
Thanks

Jason Brownlee October 21, 2019 at 6:17 am

Happy to hear that.

Perhaps you can refit your model as you get acces

Ajoy October 20, 2019 at 7:51 pm #

REPLY

Jason you know what, for “17” it returned “7”. Thats actually good, but if you could please tip me as to how can I turn this into “Awesome.” Can I divide the pixel data into 2 dividions based on whitespace or what would be the best way please to recognize “17”. Thanks

Jason Brownlee October 21, 2019 at 6:17 am #

REPLY

I believe the model supports one character at a time, try splitting multiple characters up into single character input.

Jubayed November 26, 2019 at 10:13 pm #

REPLY

Great!! I enjoyed very well.

Jason Brownlee November 27, 2019 at 6:05

Thanks!

Victor Johns November 27, 2019 at 3:43 am #

REPLY

Jason,
Thanks much for this tutorial. I had a question on the structure of the CNN. In this (and other descriptions of this problem for the MNIST digit data set) a common structure seems to be that the number of CNN filters is 32. Where each filter is a 5×5 of stride 1. However (and I apologize if this is explained somewhere) I cannot seem to find why 32 filters are chosen. i.e. Why not 16 or 64 (or 14 or 28). Is it possible to explain why this is so.

Regards
Victor

Jason Brownlee November 27, 2019 at 6:13

It is arbitrary.

Try different numbers of filters and compare the re

Samuel December 11, 2019 at 8:36 am #

REPLY

Jason,
Thank you for the detailed explanation of this. I wanted to run your code and follow along to see how it worked. When I tried to run the complete example for the baseline cnn model for mnist, I get the following error:

KeyError Traceback (most recent call last)
in
107
108 # entry point, run the test harness
–> 109 run_test_harness()

in run_test_harness()
102 scores, histories = evaluate_model(model, trainX, trainY)
103 # learning curves
–> 104 summarize_diagnostics(histories)
105 # summarize estimated performance
106 summarize_performance(scores)

in summarize_diagnostics(histories)
79 pyplot.subplot(212)
80 pyplot.title('Classification Accuracy')
—> 81 pyplot.plot(histories[i].history['acc'], color='blue'

82 pyplot.plot(histories[i].history['val_acc'], color='orange', label='test')
83 pyplot.show()

KeyError: 'acc'

Any ideas what it means?

REPLY

Jason Brownlee December 11, 2019 at 1:41 pm #

Yes, the API has changed.

I have updated the example.

Thanks for letting me know!

Xenon December 12, 2019 at 5:16 am #

please how can i Train a simple convolutional
that's what i want to see.

Jason Brownlee December 12, 2019 at 6:33 am #

See the above tutorial for an example.

REPLY

Vivek December 23, 2019 at 10:30 pm #

I want to create a image exactly same as mnist using mspaint for example and test it using the mnist evaluation model using predict_classes API.

Is it possible without using python API.

REPLY

Jason Brownlee December 24, 2019 at 6:42 am #

I don't see why not.

REPLY

Alexander Soare December 27, 2019 at 8:00 am #

Hi Jason,
Thanks for the excellent tutorial! I have two questions a

introduction" article you say that the model is discarded each time we switch the hold out set. So here are my questions

1) Here it looks like you don't discard the model between iterations of the k-fold cross validation. Why?

2) The model's accuracy gets better each time you do an iteration. Why is taking a simple average fair then? To take it to the extreme: What if you only trained on 1 epoch? Then you'd get 94% on your first fold, then 98.5% on your second fold. And so on, yielding a much poorer average and a huge standard deviation. On the back of this question, I'd also ask why the standard deviation is a fair representation of the model under these circumstances.

Thanks again for the great material!
Alex

Jason Brownlee December 28, 2019 at 7:38

You're welcome.

That looks like an error. I will update the tutorial.

Thanks for point it out!

Alexander Soare December 29, 2019 at 5:43 am #

Thanks for picking up my previous question Jason. I have another. You say at some point that there are no clear signs of overfitting. But it looks like the orange lines for the validation accuracy are staying relatively low compared to the blue lines for the testing accuracy. Is this not an indicator of overfitting? Or am I misinterpreting the charts?

REPLY

Jason Brownlee December 29, 2019 at 6:09 am #

Excellent question Alexander.

They are close enough that I would not classify it as overfitting, instead it is a good fit:
https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

REPLY

Anja January 3, 2020 at 3:02 pm #

Hello Jason,

When I execute the final code I get following error: "AttributeError: module 'tensorflow' has no attribute 'get_default_graph". I have installed Tensorflow 2.0, wi

problem?

Thank you for your time and this amazing post!

REPLY

Jason Brownlee January 4, 2020 at 8:24 am #

Confirm you have Keras 2.3 and Tensorflow 2.0 installed.

REPLY

Anja January 6, 2020 at 12:51 pm #

Hello Jason,

I have Tensorflow 2.0 and Keras 2.2.4. I tried u
Windows 10. I tried also switching to Linux on
you have any advice for me, I would be really g

Jason Brownlee January 6, 2020 a

You can try:

```
pip install keras
```

Or, if you are on anaconda:

```
conda install -c conda-forge keras
```

REPLY

Dmytro February 4, 2020 at 7:44 am #

Hello Jason,
Thank you for the tutorial and all the answers!

I have my own set of images, each class in a separate folder. What should I do to upload the dataset instead of writing

load dataset
(trainX, trainY), (testX, testY) = mnist.load_data()

It would be great if you can give some advice!

Jason Brownlee February 4, 2020 at 8:02 am

Here is an example:
https://machinelearningmastery.com/how-to-load-large-datasets-from-directories-for-deep-learning-with-keras/

Dmytro February 5, 2020 at 4:00 pm #

REPLY

Thank you!

Jason Brownlee February 6, 2020 at 8:18 am

REPLY

You're welcome.

SARVANI CH March 14, 2020 at 9:11 pm #

Hi Jason,
Thanks for the article!!!
Do you have any plans to implement "Information bottl
information among various layers in an architecture?
Does it seem simpler or a harder one for you??

Jason Brownlee March 15, 2020 at 6:13 am #

REPLY

I have not heard of it sorry, do you have a link?

SWETHA March 20, 2020 at 1:05 pm #

REPLY

def load_image(filename):
# load the image
img = load_img(filename, grayscale=True, target_size=(28, 28))
# convert to array
img = img_to_array(img)
# reshape into a single sample with 1 channel
img = img.reshape(1, 28, 28, 1)
# prepare pixel data
img = img.astype('float32')
img = img / 255.0
return img

what does the ‘filename’ mean in the above code?

REPLY

Jason Brownlee March 20, 2020 at 1:20 pm #

It’s an argument to the function – the name of the file you want to load.

REPLY

SWETHA March 20, 2020 at 2:17 pm #

Did you mean the saved model’s filename?

Jason Brownlee March 21, 2020 at

No, the function loads an image a

SWETHA March 20, 2020 at 1:48 pm #

Thank you!

REPLY

Jason Brownlee March 21, 2020 at 8:16 am #

You’re welcome.

REPLY

Elbhednam March 26, 2020 at 6:52 am #

What is the purpose of reshaping the data to a single color channel? Is it necessary for the 2D convolution step in the model definition? I ask because other digit classification examples I’ve looked at go straight to flattening the train/test data but those didn’t have a 2D convolution layer.

REPLY

Jason Brownlee March 26, 2020 at 8:07 am #

The CNN layer expects data to have a 3d shape including a channels dimension:
https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/

Edward April 13, 2020 at 5:00 am #

REPLY

If one wanted to create a new MNIST how would they set the data up? In these examples we get to use curated datasets from a dispensary but that does not explain how to set up your data to run the model. For example what if I wanted to create my own MNIST to determine if the picture is a poker card, how would I go about setting the pictures and telling the machine what that picture is?

Jason Brownlee April 13, 2020 at 6:22 am #

REPLY

Good question, see this:
https://machinelearningmastery.com/how-to-load-la
keras/

Amit Baghel April 15, 2020 at 3:53 pm #

hi Jason Brownlee,
i want to make model handwriting alphbet so please su

Jason Brownlee April 16, 2020 at 5:57 am #

Perhaps this will help:
https://machinelearningmastery.com/faq/single-faq/where-can-i-get-a-dataset-on-___

Nawal April 17, 2020 at 6:03 pm #

REPLY

Jason, Thank you so much for sharing your knowledge. I just started learning python, tensorflow and machine learning. I understood all the maths and processes but was having difficulty in coding. Your example worked first time. Brilliant.
Do you have a simpler example of CNN with just 4 hidden layers as follows:
1. Input layer of 784 nodes
2. First convolution later : 5x5x32
3. First max pooling layer
4. Second convolution layer 5x5x64
5. Second max pooling layer
6. Out put layer of 10 nodes

Jason Brownlee April 18, 2020 at 5:42 am #

Thanks.

You can adapt the above example to this directly if you like.

REPLY

Joyce May 4, 2020 at 6:03 am #

Hi Jason. Can I ask why did you reshape the data? The image from the MNIST are already grey scale which are 1 channel. I am not quite understand why you need to do this step. Also what does the trainx.shape[0]/testX.shape[0] do here?

```
# reshape dataset to have a single channel
trainX = trainX.reshape((trainX.shape[0], 28, 28, 1))
testX = testX.reshape((testX.shape[0], 28, 28, 1))
```


Jason Brownlee May 4, 2020 at 6:28 am #

Inputs to CNNs must have the channel de
grayscale images.

Amy May 4, 2020 at 6:30 am #

Can I ask why the hot encode is 10 binary vectors? Why choose 10?

REPLY

Jason Brownlee May 4, 2020 at 6:33 am #

Because there are 10 classes.

REPLY

Ruslaniv May 7, 2020 at 9:20 pm #

Hi Jason! Thank you so much, this is a great tutorial!
Just one question, i'm somewhat confused.
Are training our model twice? Once with k-fold validation and then right before saving it, with model.fit?

```
====
model = define_model()
model.fit(trainX, trainY, epochs=10, batch_size=32, verbose=0)
model.save('final_model.h5')
====
```

Jason Brownlee May 8, 2020 at 6:31 am #

REPLY

Cross-validation will train k models and is used to estimate the performance when making predictions on new data. These models are discarded.

Once we choose a final model config, we fit a model on all data and use it to start making predictions.

Ruslaniv May 11, 2020 at 8:20 pm #

REPLY

Great, that's what I thought! Thank you again for this amazing tutorial.

Jason Brownlee May 12, 2020 at 6

You're welcome.

Thenerd February 5, 2021 at 3:05 am #

In the k fold cross validation a new m
a new model is trained each time? But then wh

By model config you mean an optimal define_model? And once an optimal model is fixed we train on the entire training set? I am not able to relate how will K iterations in the K-fold reflect our understanding of what model configuration is the best.

Jason Brownlee February 5, 2021 at 5:46 am #

REPLY

Yes, a new model is trained each iteration, evaluated, and discarded.

The purpose is to estimate the performance of the model configuration when making predictions on unseen data for your prediction problem.

Once we have this estimate, we can choose to use the configuration (compared to other configurations). A final model can be fit on all data and we can start making predictions for real on new data where we don't know the label.

Thenerd February 10, 2021 at 7:21 am #

Thankyou very much. I had another question. I was trying to run the final model on the unseen test data. however for eac

run this over 10 runs and average the test results, something like monte carlo simulation. but again in that case where would model = define_model() be places? would that be placed inside the for loop for average or just be defined once before running these 10 iterations, considering this average results is on the test data. since in kfold you discard it.

Jason Brownlee February 10, 2021 at 8:13 am #

You're welcome.

Yes, you can reduce variance in prediction by fitting multiple final models and averaging their predictions:
https://machinelearningmastery.com/h

This has a code example:
https://machinelearningmastery.com/m
networks/

Gohel vivek May 11, 2020 at 6:02 pm #

If i am writing this all by making a class name

AttributeError: 'mnist_classification' object has no attrib

this .astype error is coming please sir can you help me to figure this out

REPLY

Jason Brownlee May 12, 2020 at 6:40 am #

Sorry to hear that, this will help:
https://machinelearningmastery.com/faq/single-faq/why-does-the-code-in-the-tutorial-not-work-for-me

REPLY

Ramya June 2, 2020 at 7:01 am #

Hi Jason,

Thanks for this tutorial. It's very helpful.

REPLY

Jason Brownlee June 2, 2020 at 7:55 am #

You're welcome, I'm happy to hear that.

Milind Naidu June 21, 2020 at 11:28 pm #

REPLY

Hi, i am trying to build a CNN fo the same problem. My output layer has 10 neurons. When i try to fit my model after One-Hot_encoding of y_train and y_test, i get an error that the labels and the logits don't match. However, without the one-hot encoding the model fits perfectly. Can you please explain?

Jason Brownlee June 22, 2020 at 6:14 am #

REPLY

Perhaps you can use the above tutorial as a starting point and adapt it for your project?

William July 19, 2020 at 8:48 pm #

This is the best article about MNIST, but I hav
performance in order to running these codes?"

Jason Brownlee July 20, 2020 at 6:12 am #

Thanks!

No, you can run on CPU, it might just take more minutes to run.

Sandhya July 31, 2020 at 3:58 pm #

REPLY

Hi Jason
Thanks for this amazing tutorial. It will definitely help a lot of budding researchers.
I have a question
Can you please help if instead of single character i have to work on complete self taken handwritten word image (20 different classes,total around 20000 images). How to do it?

Jason Brownlee August 1, 2020 at 6:07 am #

REPLY

Perhaps the letters in each word images can first be segmented, then classified.

Sandhya August 1, 2020 at 5:39 pm #

REPLY

thanks for the reply but i want to do it

Jason Brownlee August 2, 2020 at 5:39 am #

REPLY

Perhaps use a CNN to read the images and an LSTM to interpret the image features and output one letter at a time.

Sandhya July 31, 2020 at 4:14 pm #

REPLY

hi Jason
Thanks for this amazing tutorial.

Jason Brownlee August 1, 2020 at 6:07 am #

You're welcome.

zulfiqar ali October 18, 2020 at 9:46 am #

Thanks, Jason Brownlee for this excellent tutorial. I am your huge fan and I regularly following your all machine learning and deep earning models.I got a lot of knowledge from this site but After completion of machine learning tutorials there always raise a question in my mind that how can we apply these models in Mobile APPs(Andriod, iOS, etc) so that we get more fun and knowledge by this models. Are you suggest to me any link to have such kind of projects such as weather prediction app . Thanks again.

Jason Brownlee October 18, 2020 at 1:25 pm #

REPLY

Thanks.

Sorry, I don't know about using models on mobile.

Zunaira Shafqat October 23, 2020 at 4:49 am #

REPLY

Thanks for the tutorial Jason Brownlee.
Can I use same model but pass my own dataset (of humans) to it & use it for detecting fall or not fall by making small changes?
Thanks.

Jason Brownlee October 23, 2020 at 6:17 am #

REPLY

Sure.

Preethi November 12, 2020 at 3:28 pm #

REPLY

Hi do you have any projects done based on Wireless and Mobile Network except human activity recognition using smart phone

Jason Brownlee November 13, 2020 at 6:32

I don't think so. Perhaps try the blog sear

Archna November 13, 2020 at 7:01 pm #

Thanks for the detailed explanation !!
Can you pls do a tutorial on colorization without using

Thanks in advance

Jason Brownlee November 14, 2020 at 6:30 am #

REPLY

Great suggestion, thanks!

Archna November 24, 2020 at 6:31 pm #

REPLY

Waiting for the same ☺

Ebdulmomen December 7, 2020 at 8:53 am #

REPLY

hey jason that was a very good tutorial, i did not understand how in the predict_classes we got 7? the model had a softmax activation function as the last layer which will be a probability distribution of 10 float numbers, what i can't grasp is how did we get back 7?

Ebdulmomen December 7, 2020 at 8:57 am #

REPLY

what i mean is how this 10 probabilities turned into 1 predicted number?

Ebdulmomen December 7, 2020 at 9:09 am #

REPLY

and how does it know it is 7, didn't we hot encode the Y values?

Jason Brownlee December 7, 2020 at 1

Via argmax:
https://machinelearningmastery.com/argmax-in

Jason Brownlee December 7, 2020 at 1:35 p

Thanks.

The predict_classes() performs an argmax on the

If you are new to argmax, see this:
https://machinelearningmastery.com/argmax-in-machine-learning/

Ebdulmomen December 7, 2020 at 6:41 pm #

REPLY

thank you jason, everything is clear, really appreciate it!

Jason Brownlee December 8, 2020 at 7:40 am #

REPLY

I'm happy to hear that.

Naveen December 13, 2020 at 2:43 am #

REPLY

Hey is this code not compatible for Tensorflow 2? I am getting the following error.
E tensorflow/stream_executor/cuda/cuda_driver.cc:314] failed call to cuInit: CUDA_ERROR_NO_DEVICE: no CUDA-capable device is detected

Jason Brownlee December 13, 2020 at 6:06 am #

REPLY

Yes, you can learn more here:
https://machinelearningmastery.com/faq/single-faq/do-you-support-tensorflow-2

That error looks like a problem with your development environment.

Yash December 23, 2020 at 9:40 am #

REPLY

Hello!
Thanks for the wonderful article helped me a lot with m
I have a question though, I am trying to take an image
number but it doesn't work with that. What should I do
I resized it to 28×28 using online tools and even tried t
Any suggestion or lead will be appreciated!
Thanks in advance!

Jason Brownlee December 23, 2020 at 1:27

You will have to prepare the image in an i
white foreground, black background, grayscale and same image size.

JG December 31, 2020 at 7:53 am #

REPLY

Hi Jason,

Great tutorial as always!

I implemented your tutorial with two variants.

I load the images from digits dataset of sklearn library (it uses less images number only 1,797 and with much lower resolution 8 x 8 bits vs MNIST 70,000 and 28,28 pixels resolution. On the contrary it is much faster this dataset because it use less data features.
And also I apply for this multi-class problem other library classifier models from Sklearn such as SVC(), logisticRegression(), ExtraTreesClassifier(), XGBClassifier(), RandomForestClassifier(), etc.

My main results are:

– I got 98.4% accuracy and 1.1 of sigma for the simple CNN baseline model (not other deep layers, not other BatchNormalization()) and I got 98.8% from SVC…but only suing the 8×8 pixels resolution which is a great new!.

But when I reduce the final image to be predicted to 8 x 8 pixels, in order to apply my trained model, where I got such a great score I poorly predict the 7 digit as 9 c

My guess is when I load_image of 7 and I clip it to such smaller size of 8×8 pixels, I lost important images features in the process of cutting back the image….where the the MNIST 28×28 pixels still retain key digit features of the image, what do you think?

thank you Jason

REPLY

Jason Brownlee December 31, 2020 at 9:26 am #

Thanks.

Very nice experiments!

Yes, the larger images perhaps provide more cont

Hannes January 26, 2021 at 1:15 am #

Hi Jason,

You have always very interesting articles.
I was able to train a CNN based on the Char74K-datas
42MB and the time to predict a number is fast.

I did use a kNN-model before (on the same dataset) th

Thanks,
Hannes

REPLY

Jason Brownlee January 26, 2021 at 5:58 am #

Thanks.

Well done!

REPLY

Mukesh February 5, 2021 at 8:42 pm #

Hi Jason,

Great article!!

i just have one doubt, in the evaluate method why are we creating a new model for every fold?

Thanks,

Jason Brownlee February 6, 2021 at 5:49 am #

REPLY

This is how k-fold cross-validation works, we have a new training set and fit a new model for each fold, then average the performance of all models to get an estimate of model performance when used on unseen data.

You can learn more here:
https://machinelearningmastery.com/k-fold-cross-validation/

Dmitry February 6, 2021 at 6:20 pm #

REPLY

Thanks for this amazing tutorial!

I have some experience in MLP but not with CNN. Is th
data have shape 16×1 (row) with values between 0,01
for 15 classes. If I reshape samples from 16×1 to 4×4 f
shoud I use? Or better dont reshape and try model witl

Jason Brownlee February 7, 2021 at 5:17 am

CNN can be effective if there is a spatial relationship between the inputs in an image or a sequence.

If you have tabular data (e.g. not a images and not sequences), then a CNN does not make sense.

Ahmed Shafeek February 18, 2021 at 2:19 pm #

REPLY

Hi Jason,

Thank you very much for your amazing tutorial. I am running your code using pycharm and I have a GPU with the right CUDA but your code is only running on my CPU.

Is there something that I am missing here?

thank you. ☺

Ahmed Shafeek February 18, 2021 at 4:52 pm #

REPLY

Sorry Jason, it turned out I had a problem in the CuDNN installation and i did a work around to fix it and it worked. ☺

Thanks again for this great tutorial. ☺

Jason Brownlee February 19, 2021 at 5:55 am #

REPLY

No problem!

Happy to hear you fixed the issue.

Jason Brownlee February 19, 2021 at 5:54 am #

REPLY

The code is agnostic to hardware – runs on both.

If it is running on your CPU, then you need to char
have tutorials on this topic, sorry.

Karl February 24, 2021 at 1:20 am #

Hi Jason,

Thank you very much for this tutorial, helped a lot! I de
well according to the mean accuracy. However, the cro
of the training set. Is this a sign of underfitting? And if s
the model to unseen data?

Jason Brownlee February 24, 2021 at 5:34 am #

REPLY

You're welcome.

Well done!

If performance is poor, a learning curve can help diagnose issues, this can help you interpret the learning curve:
https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

Noah February 24, 2021 at 1:51 am #

REPLY

Hi Jason,

Thank you very much for this tutorial, helped a lot! I developed my own CNN, which seems to perform pretty well according to the mean accuracy. However, the cross-entropy loss of the validation set is below the loss of the training set. Is this a sign of underfitting? And if s
the model to unseen data?

Jason Brownlee February 24, 2021 at 5:35 am #

REPLY

You're welcome.

Perhaps focus on the out of sample performance of the model first and optimize that.

Geo March 2, 2021 at 12:20 am #

REPLY

Hi Jason,

Previously I was grateful for the tutorial you provided, I
probability value of the prediction result? (for example

Jason Brownlee March 2, 2021 at 5:45 am #

Thanks!

You can call model.predict() then multiply the resul

William March 2, 2021 at 3:01 pm #

REPLY

Thank you for the tutorial!

I created an image with same characteristics as sample_image.png (1490×1480, black background, white foreground), opened it in Paint, typed a "6" (without the quotes, of course), increased the font size to 1000 (so it would occupy most of the canvas), then saved it. However, the code doesn't seem to recognize it. It seems to think it's a "1", "8", or other numbers.

Any suggestions on why?

Jason Brownlee March 3, 2021 at 5:25 am #

REPLY

Perhaps there was some important difference in the image itself or it's preparation (e.g. pixel scaling) that differed from the training dataset?

Venkat April 1, 2021 at 10:07 pm #

REPLY

Hi Jason – fantastic article!! Just wondering why you had to do evaluate to get the accuracy, while it is already being returned by the fit call? I just checked the histories and they have exactly the same stuff returned by the evaluate call. Was there any other reason for the evaluate call?

REPLY

Jason Brownlee April 2, 2021 at 5:39 am #

Thanks!

No, accuracy on a hold out dataset is not calculated when calling fit. We must make predictions on new data manually or use the evaluate function.

Venkat April 15, 2021 at 4:17 am #

Thanks for the reply!! I was seeing the
evaluate calls for validation. As a result, the re
returned by the evaluate call. Am I making sen

```
# fit model
history = model.fit(trainX, trainY, epochs=10, b
verbose=0)
# evaluate model
_, acc = model.evaluate(testX, testY, verbose=0)
```


REPLY

Jason Brownlee April 15, 2021 at 5:30 am #

Generally it is not a good idea to use test data as validation, I do to keep the examples simple.

REPLY

Berke April 4, 2021 at 9:47 pm #

Hi Jason, thanks for this great tutorial!

I don't get why we don't use the models that we created during the k-fold. I was expecting something like we combine this k models results into one single model and save it. If we don't do that how's that useful?

REPLY

Jason Brownlee April 5, 2021 at 6:11 am #

Models created during k-fold cross-validation are discarded. They are only used to estimate the performance of the model/pipeline on unseen exar

Once we choose a configuration, we can fit a final model on all data and use it to make predictions on new examples:
https://machinelearningmastery.com/train-final-machine-learning-model/

REPLY

Vishwa April 15, 2021 at 9:38 pm #

Thanks a lot Jason !!! This tutorial was very very helpful and it solved half of my project problems !!! Thanks a lot again !..

Jason Brownlee April 16, 2021 at 5:30 am #

You're welcome.

ismail June 9, 2021 at 8:47 am #

Thank you for the great tutorial. Is it possible t

REPLY

Jason Brownlee June 10, 2021 at 5:20 am #

Perhaps try it and see.

REPLY

Taylor Rayne June 12, 2021 at 12:20 pm #

Hello! Thank you Jason for this amazing resource. I am currently working on a rented computer from school – do you think I could do implement this code in CoLabs?

REPLY

Jason Brownlee June 13, 2021 at 5:46 am #

You're welcome.

Good question, see this:
https://machinelearningmastery.com/faq/single-faq/do-code-examples-run-on-google-colab

Manal June 15, 2021 at 4:14 pm #

Hi Jason,
Thank you very much
If possible, advise on the implementation HMM.

REPLY

Jason Brownlee June 16, 2021 at 6:17 am #

Thanks for the suggestion, perhaps in the future.

REPLY

amberbir June 15, 2021 at 11:11 pm #

i am developing a model of handwritten chara
is not only because i have a small datatset, there is so
improvement

Jason Brownlee June 16, 2021 at 6:21 am #

Perhaps try some of these suggestions:
https://machinelearningmastery.com/improve-deep

REPLY

Shri July 8, 2021 at 11:15 pm #

Can I use this code for any character dataset ?

REPLY

Jason Brownlee July 9, 2021 at 5:11 am #

Perhaps try it and see.

REPLY

Shireesh Apte August 8, 2021 at 8:34 am #

Could you provide a peer review for a manuscript that has been submitted to the Journal of High School Science (http://jhss.scholasticahq.com) ? The title of the manuscript is : Implementing a Quantum Convolutional Neural Network for Efficient Image Recognition. Abstract: Machine learning has many real-world applications ranging from modeling the universe to computational chemistry. As probability is the bedrock for machine learning, it is essential to optimize both hardware and software to obtain the best results. Classical computers are generally used for machine learning programs. However, learning from high-dimensional data often demands excessive comp

highest accuracy. The Quantum Computing environment can be utilized to create a more accurate model than that created via classical computing. To test this quantum advantage, we implemented a Quantum Convolutional Neural Network (QCNN), which parallels the structure of the classical Convolution Neural Network (CNN) in the quantum domain. Due to the lack of quantum computers with many qubits, physicists, namely John Preskill, have introduced the Noisy Intermediate Scale Quantum (NISQ) concept, which constitutes a hybrid interface between classical and quantum computers. In the context of QCNN, the data processing and the cost function optimization would be performed on the classical computer, while the probabilities generated by the Variational Quantum Circuits (VQC) would be evaluated on the quantum computer. The QCNN consists of a classical-to-quantum data encoder, a cluster state quantum circuit to entangle qubit states, a series of Variational Quantum Circuits using Quantum Convolutional and Pooling Layers for efficient feature extraction, a quantum-to-classical data decoder, which would lead to the output. Both the CNN and the QCNN extract features from data like 2D images, and performances can be compared using metrics like accuracy and time.

Jason Brownlee August 9, 2021 at 5:52 am #

Sorry I cannot peer review your manuscri

Leave a Reply

Name (required)

Email (will not be published) (required)

Website

SUBMIT COMMENT

Welcome!
I'm *Jason Brownlee* PhD
and I **help developers** get results with **machine learning**.
Read more

Never miss a tutorial:

Picked for you:

How to Train an Object Detection Model with Keras

How to Develop a Face Recognition System Using FaceNet

How to Perform Object Detection With YOLOv3 in Keras

How to Classify Photos of Dogs and Cats (with 97% accurac

How to Get Started With Deep Learning for Computer Vision

Loving the Tutorials?

The Deep Learning for Computer Vision EBook is where you'll find the ***Really Good*** stuff.

>> SEE WHAT'S INSIDE

LinkedIn | Twitter | Facebook | Newsletter | RSS

Privacy | Disclaimer | Terms | Contact | Sitemap | Search

Start Machine Learning