Illinois Institute of Technology

# ECE 566 Machine and Deep Learning

# Project 2

**Pranati Trivedi**

**A20424327**

**Due date: 11/30/2019**

**List of figures:**

Figure 1. Model Design in which input is Input image and output is classes of 10 digits

Figure 2. Model loss

Figure 3. Model accuracy

Figure 4. Loss graph when overfitting due to less convolution layer and not using dropout layer

Figure 5. Accuracy graph when overfitting due to less convolution layer and not using dropout layer

Figure 6. Example of handwritten digit with correlated noise

Figure 7. CNN layers and its output size

Figure 8. Example of model summary with loss and accuracy

Figure 9. Test Accuracy

**Goal:** To design, train and test a Convolutional Neural Network (CNN) multi-class classification problem.

Experiment:

Here in this project, we are using "MNIST handwritten digit database" with additive correlated noise. To design and train a convolutional neural network that classifies a noisy handwritten digit into one of the ten possible classes. For that the python libraries used are 'matplot inline' to plot the graphs under the notebook commands, 'os module' to interact with os dependent functionality to interact with file system, 'pickel' library to serialize the matrix and to convert a python object into a character stream. Keras library for different sequence of dataset, its callbacks to get the information of its internal states and statistics of model during the training, optimizer to compile the keras model. As we know for convolution we need different layers to get the weights so keras's layers module provide different layers to make a training more accurate.

There are different two models in Keras, Sequential and model used with the functional API. These models have different methods, here we used sequential model. Then the loss function of keras to return the scalar for each data point and takes y_true and y_pred and give the mean of the output arrays across all data points.

After mounting the dataset in google drive folder, dividing the train data in training data and its label in y_train variable. Different unique 10 classes in num_classes variable. Reshape the training data and put the classes in binary representation using 'categorical' method.

As here we are using CNN model, then keras's function to save weights data in HDF5 file.

As we know Hyper parameters are really important in machine learning to train the model. Main three parameters are:

1. Batch size: This is a hyperparameter that define the number of samples to work through before updating the internal model parameters. Here I have evenly divided training data in set of 7 samples. Predictions are compared to the expected output variables and error is calculated. From the error, the updated algorithm is used to improve the model, move down along with error gradient. I have used the mini-batch gradient descent algorithm.
2. Epoch: This hyperparameter, that defines the number times that learning algorithm will work through the entire training set. One epoch means each sample in the training dataset has had an opportunity to update the internal model parameters such as weights and bias. An epoch is comprised in of one or more batches. The number of epochs, allowing the learning algorithm to learn until the error from the model has been sufficiently minimized. After setting the epoch as 22, we can make the learning curve and see the model is over learned, under learned or fit for the training dataset.
3. Validation split: Validation split is the amount of validation data of 0.1% and train data of 0.9%. The first set is used for training and the $2^{nd}$ set for validation after each epoch.

**CNN model design:**

CNN design is of feature extraction and finishes with classification. Feature extraction is performed by alternating convolution layers with subsampling layers. Classification performed with dense layers followed by a final softmax layer.

To train the model I have used 3 features:

The double convolution layer trick, only learnable subsampling layer trick, batch normalization and achieves the best accuracy. Here I have used 40% dropout method to overcome the problem of overfitting. And converting kernel size of 5 to two 3 to increase the accuracy.

*Convolution layer :*

32 filters are used, convolution kernel is a size of 3x3, the new layer maps will have a size equal to the new layer maps divided by strides. Padding is valid by default so the size of the new layer maps is reduced by kernel size 1. Activation is applied during forward propagation.

*Batch Normalization :*

As the data flows through a deep network, the weights and parameters adjust those values, sometimes making the data too big or too small again - a problem the authors refer to as "internal covariate shift". By normalizing the data in each mini-batch, this problem is largely avoided. Batch Normalization normalizes each batch by both mean and variance reference. Networks train faster converge much more quickly. Allows higher learning rates. Gradient descent usually requires small learning rates for the network to converge. Makes weights easier to initialize. Makes more activation functions viable. Because batch normalization regulates the values going into each activation function, non-linearities that don't seem to work well in deep networks actually become viable again. May give better results overall.

*Flatten* is the function that converts the pooled feature map to a single column that is passed to the fully connected layer. *Dense* adds the fully connected layer to the neural network.

Using model.summary(),we can view all of the layer sizes in your convolutional network.

*Model.compile* is used for model to training. List of metrics to be evaluated by the model during training and testing. The loss value that will be minimized by the model will then be the sum of all individual losses.

Optimizer: To model the training optimizer is an algorithm which updates the weight parameters to minimize the loss function acts as a guides to the terrain telling optimizer if it is moving in the right direction to reach the bottom of the valley, the global minimum. I have used Adam optimizer which is an adaptive learning rate optimization algorithm that's been design specifically for training deep neural networks. The algorithms leverages the power of adaptive learning rates methods to find individual learning rates for each parameter.

`Model.fit` function is used to train the model for fixed number of epochs (iteration). In which we will give x input data, y target data, epochs, batch size, verbose as 1 as progress bar, callbacks, validation split and shuffle to shuffle the training data before each epoch.

*Callback:* I have used `ReduceLROnPlateau`. In which reduced learning rate when a metric has stopped improving. Models often benefit from reducing the learning rate by a factor of 2-10 once learning stagnates. This callback monitors a quantity and if no improvement is seen for a 'patience' number of epochs, the learning rate is reduced.

As we can see in the figure 1. , I have used 2 convolution layer with valid padding, 1 stride and 3x3 kernels and then 1 convolution layer of same padding, 2 strides and 5x5 kernel size. Then to remove the overfitting I have used the dropout layer with 40 % dropout. All these are with using the relu activation function. After that using 64 feature maps did the same 2 convolution layer with 3x3 kernel size with batch normalization and 1 with 5x5 kernel size and batch normalization. After that perform dropout of 40%.

For classification, I have used fully connected network using flatten layer with 1024 units, Dense layer of 128 units and batch normalization with dropout. Dense layer for 10 units which are classes of 10 digits using Softmax activation function. At the end test the model on test set data and find the accuracy.

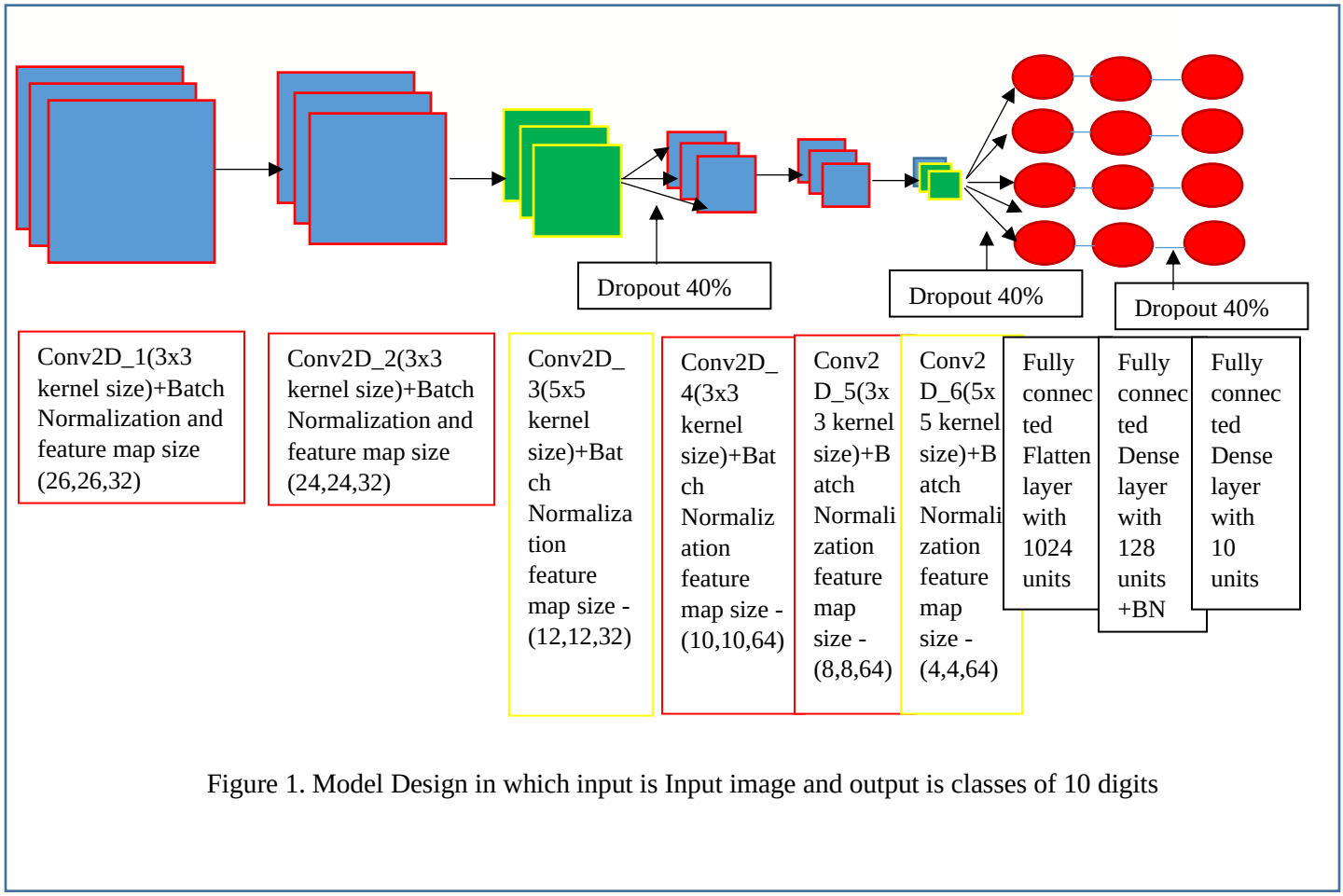



Figure 1. Model Design in which input is Input image and output is classes of 10 digits

**Training, Validation and Test accuracy:**

Train on 54000 samples and validate on 6000 samples.

I got training accuracy of **93.12%** ,validation accuracy of **95.12 %** and test accuracy is **87.21 %.**

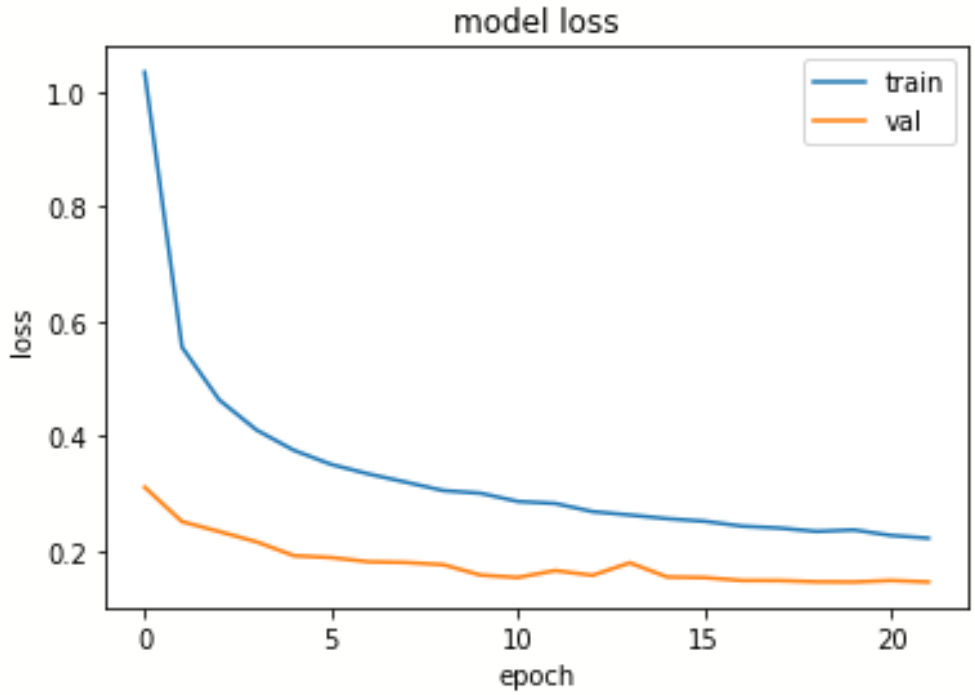



Figure 2. Model Loss

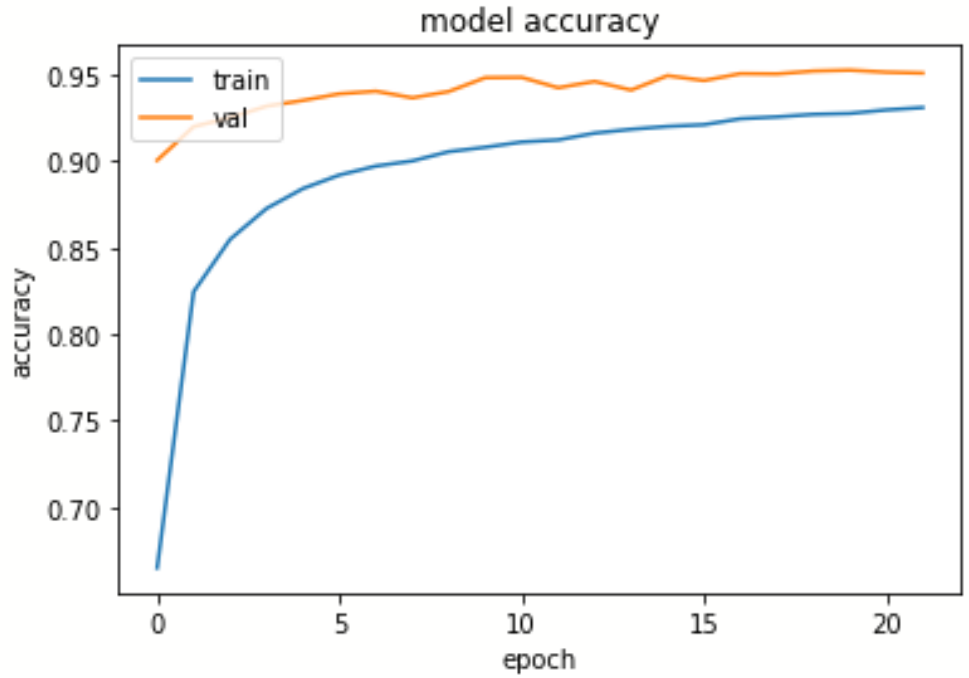



Figure 3. Model Accuracy

Total training time: **55 minutes.**

**Conclusion:**

From this experiment and its result I can say that accuracy is highly depend on the depth and type of the layers. Hyperparameters like batch size, validation split and epochs are also really important. As with only 1 convolution layer and batch size of 7, validation split of 0.1 and epochs of 22 I got accuracy around 91% as shown in Figure 5 and loss is also increases as shown in Figure 4 but after adding more layers and there was overfiiting, so I have used dropout layer. And I got validation accuracy of 95%. I got test accuracy of 87.22% because of overfitting and test the model multiple times or there can be with new data, there is possibility to get more accuracy.

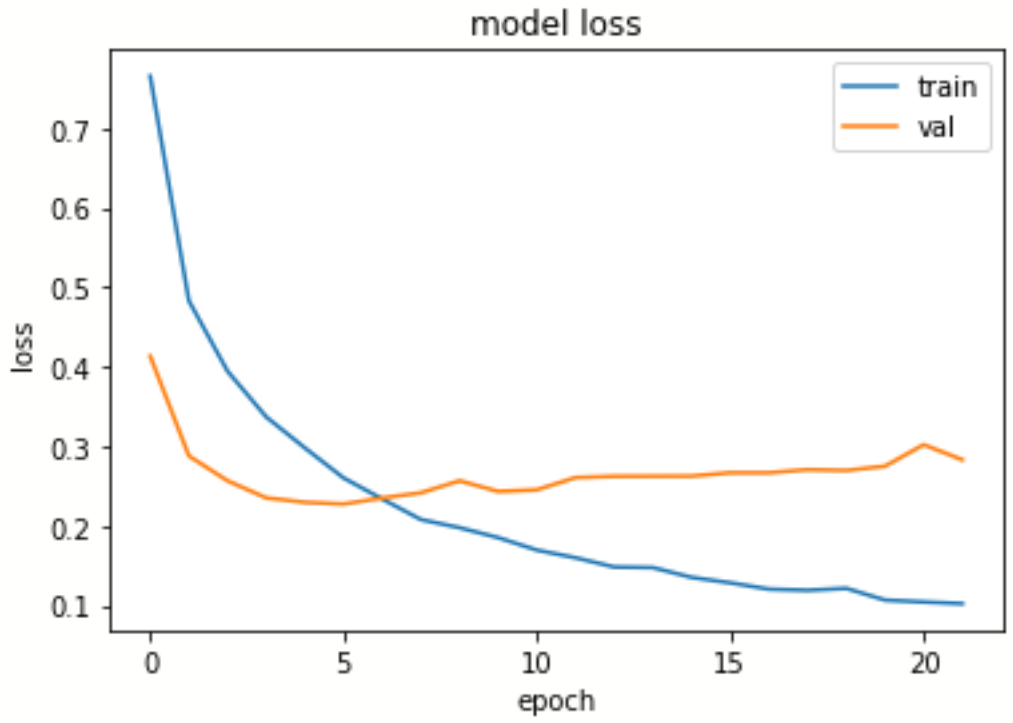



Figure 4. Loss graph when overfitting due to less convolution layer and not using dropout layer

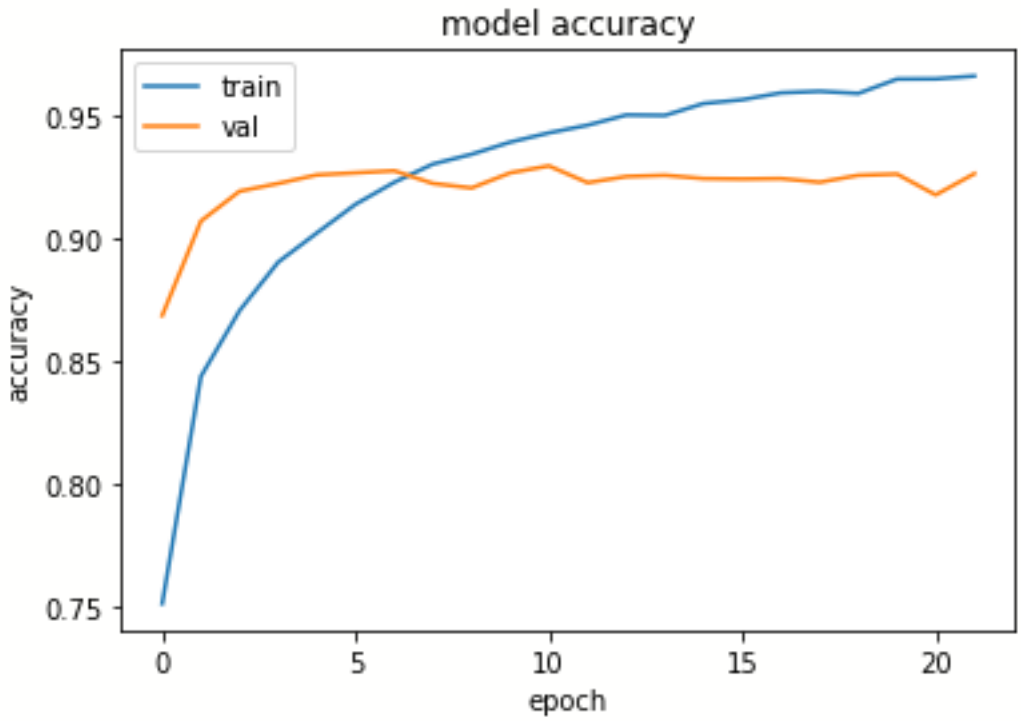



Figure 5. Accuracy graph when overfitting due to less convolution layer and not using dropout layer

**References**:

1. https://conx.readthedocs.io/en/latest/MNIST.html
2. https://keras.io/callbacks/
3. https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/

# Appendix:

### 1.Importing Python libraries and modules

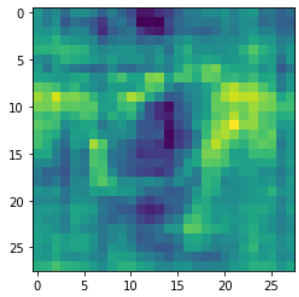
```
%matplotlib inline
import os
import numpy as np
import pickle
import matplotlib.pyplot as plt
from keras.utils import to_categorical
from keras.callbacks import ReduceLROnPlateau, ModelCheckpoint, EarlyStopp
ing
from keras.optimizers import Adadelta, Adam, SGD
from keras.layers import Input, Conv2D, Dense, MaxPooling2D, Dropout, Flat
ten, AveragePooling2D, Conv2DTranspose, UpSampling2D,BatchNormalization
from keras.models import Sequential
from keras.losses import categorical_crossentropy

from google.colab import drive
drive.mount("/content/gdrive", force_remount=True)
os.chdir("/content/gdrive/My Drive/Colab Notebooks")
```

### 2. Load and formatting the data

```
data = np.load('./MNIST_CorrNoise.npz')

x_train = data['x_train']
y_train = data['y_train']

num_cls = len(np.unique(y_train))
print('Number of classes: ' + str(num_cls))

print('Example of handwritten digit with correlated noise: \n')

k = 3000
plt.imshow(np.squeeze(x_train[k,:,:]))
plt.show()
print('Class: '+str(y_train[k])+'\n')

# RESHAPE and standarize
x_train = np.expand_dims(x_train/255,axis=3)

# convert class vectors to binary class matrices
y_train = to_categorical(y_train, num_cls)
```

```
print('Shape of x_train: '+str(x_train.shape))
print('Shape of y_train: '+str(y_train.shape))
```

```
Number of classes: 10
Example of handwritten digit with correlated noise:
```




```
Class: 9

Shape of x_train: (60000, 28, 28, 1)
Shape of y_train: (60000, 10)
```

Figure 6. Example of handwritten digit with correlated noise

## 3.Training

```
model_name='CNN' # To compare models, you can give them different names

pweight='./weights/weights_' + model_name  + '.hdf5'

if not os.path.exists('./weights'):
  os.mkdir('./weights')

## EXPLORE VALUES AND FIND A GOOD SET
b_size = 7 # batch size
val_split = 0.1 # percentage of samples used for validation (e.g. 0.5)
ep = 22 # number of epochs

input_shape = x_train.shape[1:4] #(28,28,1)

model= Sequential()

model.add(Conv2D(32,kernel_size=3,activation='relu',input_shape=(28,28,1))
)
model.add(BatchNormalization())
model.add(Conv2D(32,kernel_size=3,activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(32,kernel_size=5,strides=2,padding='same',activation='rel
u'))
model.add(BatchNormalization())
model.add(Dropout(0.4))
```

```
model.add(Conv2D(64,kernel_size=3,activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(64,kernel_size=3,activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(64,kernel_size=5,strides=2,padding='same',activation='rel
u'))
model.add(BatchNormalization())
model.add(Dropout(0.4))

model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.4))
model.add(Dense(10, activation='softmax'))

model.compile(optimizer="adam", loss="categorical_crossentropy", metrics=[
"accuracy"])
model.summary()
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2,
                              patience=5, min_lr=0.001)
callbacks_list = [reduce_lr]

history=model.fit(x_train, y_train,
                        epochs=ep,
                        batch_size=b_size,
                        #verbose=0,
                        verbose=1,
                        shuffle=True,
                        validation_split = val_split,
                        callbacks= callbacks_list)

print('CNN weights saved in ' + pweight)

# Plot loss vs epochs
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper right')
plt.show()

# Plot accuracy vs epochs
plt.plot(history.history['acc'])
```

```
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_13 (Conv2D)           (None, 26, 26, 32)        320
_________________________________________________________________
batch_normalization_15 (Batc (None, 26, 26, 32)        128
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 24, 24, 32)        9248
_________________________________________________________________
batch_normalization_16 (Batc (None, 24, 24, 32)        128
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 12, 12, 32)        25632
_________________________________________________________________
batch_normalization_17 (Batc (None, 12, 12, 32)        128
_________________________________________________________________
dropout_7 (Dropout)          (None, 12, 12, 32)        0
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 10, 10, 64)        18496
_________________________________________________________________
batch_normalization_18 (Batc (None, 10, 10, 64)        256
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 8, 8, 64)          36928
_________________________________________________________________
batch_normalization_19 (Batc (None, 8, 8, 64)          256
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 4, 4, 64)          102464
_________________________________________________________________
batch_normalization_20 (Batc (None, 4, 4, 64)          256
_________________________________________________________________
dropout_8 (Dropout)          (None, 4, 4, 64)          0
_________________________________________________________________
flatten_3 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_5 (Dense)              (None, 128)               131200
_________________________________________________________________
batch_normalization_21 (Batc (None, 128)               512
_________________________________________________________________
dropout_9 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_6 (Dense)              (None, 10)                1290
=================================================================
Total params: 327,242
Trainable params: 326,410
Non-trainable params: 832
```

Figure 7. CNN layers and its output size

```
Epoch 20/22
54000/54000 [==============================] - 149s 3ms/step - loss: 0.2365 - acc: 0.9278 - val_loss: 0.1458 - val_acc: 0.9528
Epoch 21/22
54000/54000 [==============================] - 150s 3ms/step - loss: 0.2269 - acc: 0.9298 - val_loss: 0.1483 - val_acc: 0.9517
Epoch 22/22
54000/54000 [==============================] - 148s 3ms/step - loss: 0.2221 - acc: 0.9312 - val_loss: 0.1458 - val_acc: 0.9512
CNN weights saved in ./weights/weights_CNN.hdf5
```

Figure 8. Example of model summary with loss and accuracy

### 4.Make prediction in test set

```
from keras.models import load_model

## LOAD DATA
data = np.load('./MNIST_CorrNoise.npz')

x_test = data['x_test']
y_test = data['y_test']

num_cls = len(np.unique(y_test))
print('Number of classes: ' + str(num_cls))

# RESHAPE and standarize
x_test = np.expand_dims(x_test/255,axis=3)

print('Shape of x_train: '+str(x_test.shape)+'\n')

## Define model parameters
model_name='CNN' # To compare models, you can give them different names
pweight='./weights/weights_' + model_name  + '.hdf5'

model = load_model(pweight)
y_pred = model.predict_classes(x_test)

Acc_pred = sum(y_pred == y_test)/len(y_test)

print('Accuracy in test set is: '+str(Acc_pred))
```

```
Number of classes: 10
Shape of x_train: (10000, 28, 28, 1)

Accuracy in test set is: 0.8721
```

Figure 9. Test Accuracy

**Training code when got overfitting without dropout and not included 3 convolution layer.**

```
model_name='CNN' # To compare models, you can give them different names

pweight='./weights/weights_' + model_name  + '.hdf5'

if not os.path.exists('./weights'):
  os.mkdir('./weights')

## EXPLORE VALUES AND FIND A GOOD SET
b_size = 7 # batch size
val_split = 0.1 # percentage of samples used for validation (e.g. 0.5)
ep = 22 # number of epochs

input_shape = x_train.shape[1:4] #(28,28,1)

model= Sequential()

model.add(Conv2D(32,kernel_size=3,activation='relu',input_shape=(28,28,1))
)
model.add(BatchNormalization())
model.add(Conv2D(32,kernel_size=3,activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(32,kernel_size=5,strides=2,padding='same',activation='rel
u'))
model.add(BatchNormalization())

model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(BatchNormalization())
model.add(Dense(10, activation='softmax'))

model.compile(optimizer="adam", loss="categorical_crossentropy", metrics=[
"accuracy"])
model.summary()
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2,
                              patience=5, min_lr=0.001)
callbacks_list = [reduce_lr]

history=model.fit(x_train, y_train,
                        epochs=ep,
                        batch_size=b_size,
                        #verbose=0,
                        verbose=1,
                        shuffle=True,
```

```
                        validation_split = val_split,
                        callbacks= callbacks_list)

print('CNN weights saved in ' + pweight)

# Plot loss vs epochs
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper right')
plt.show()

# Plot accuracy vs epochs
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```