A

Report on

COMP 4900 Mini-project 3

by

Group 3-22

Group members:

Zeye Gu (Student No. 101036562)

Peng Li (Student No. 101047123)

Chengyi Song (Student No. 101033544)

COMP 4900:Introduction to Machine Learning Fall 2020

Majid Komeili, PhD.

(Carleton University)

2020-12-9

ABSTRACT

The target of this project is to train the machine to recognize articles from each given image. Each article has its own price, the machine should be able to calculate the total prices of all the articles in a given image after training. To achieve this target, we built a neural network in python. We solved the problem by developing a modified version of the VGG neural network and accomplished 87.1% accuracy on a Kaggle competition. After we finished this project, we figured out some important findings. First is that the depth of hidden layers would influence the accuracy of the result. The second is choosing as many filters as possible also helps to improve accuracy. The third is that the batch size should not be too large or too small to get an accurate result.

1 Introduction and Motivation

For this project, we need to build a neural network to recognize the articles in images. Each image in the training data has three articles, including T-shirt/top, Trouser, Pullover, Dress, and Coat. Each article has different prices from 1 to 5. Our program should be able to predict the total price of articles of each image.

By doing this project, we figured three important findings. The first one is that the depth of hidden layers would influence the accuracy of prediction. If the number of hidden layers is too small, the accuracy would be very low. The second finding is that choosing a filter can help to catch more information, which helps to improve the accuracy of prediction. The third finding is that the batch size cannot be too large or too small. If the batch size is too small, we cannot get an accurate gradient descent. If the batch size is too large, the training process would take a lot of time.

2 Dataset

2.1 Dataset Introduction and preprocessing

X_train.shape	(60000, 64, 128, 1)
X_test.shape	(10000, 64, 128, 1)
y_train.shape	(60000, 9)

Table 1: Fashion-MNIST dataset shape

Before starting the experiment, we have to read the data. We use panda to read the pickle file and CSV file. We also convert all of the data input as nump array. When checking the shape of the data, we find it is 60000x64x128. Thus, there would be 60000 pictures with pixels of 64x128, we want to reshape each picture to represent as 64x128x1(3 dimensions). Except that we also have to make some changes to the train labels, since each picture is equivalence to the value 5 to 13, we want to use an array to represent each picture and the array size should be 9. For example, 7 would represent as [0,0,1,0,0,0,0,0,0,0,0]. Thus, the training labels' shape would be 60000x9.

3 Proposed Approach

3.1 CNN model

3.1.1 Forward propagation

To build the Convolutional neural network, first of all, we have to build the model Layer-by-layer linear stacked network architecture, so we can use the sequential function, it is a build-in function in Keras. The next step is to build hidden lay in our CNN, before doing convolution we do not want to lose too much information at the benign, thus we have to do zero-padding first, then after we call conv2d function would not change our matrix to would still be 64x128. Except that, since every filter represents a filter identity, the more filter that we can have meant more information would be collected. Thus our filter would be 3x3x32, after we first convulsion, we get a matrix 64x128x32. We doing the same convolution twice and use the max-pooling method to reduce the matrix column and row to 32x64x32.We will

do the same action as we increase the filters to 64 and 128. Except that, since our information increases a lot, we do not want our data over-fitting, thus we use dropout functions to throw away part of the information. The throw rate we set as 0.5. Dropout function would be called before increasing the size of filters. After convolution, our matrix should become 8X16X128. We want to convert it to 1 dimension matrix, thus we add the Flatten function and the size should be 16384(8X16X128). This number is still too large, we use the Dense function to connect all neural unit and with Relu active function output as 512x1. Finally, since it is a 9 classification problem, we convert it using the Dense function to 9x1 output and active with soft-max function to choose the best output y.

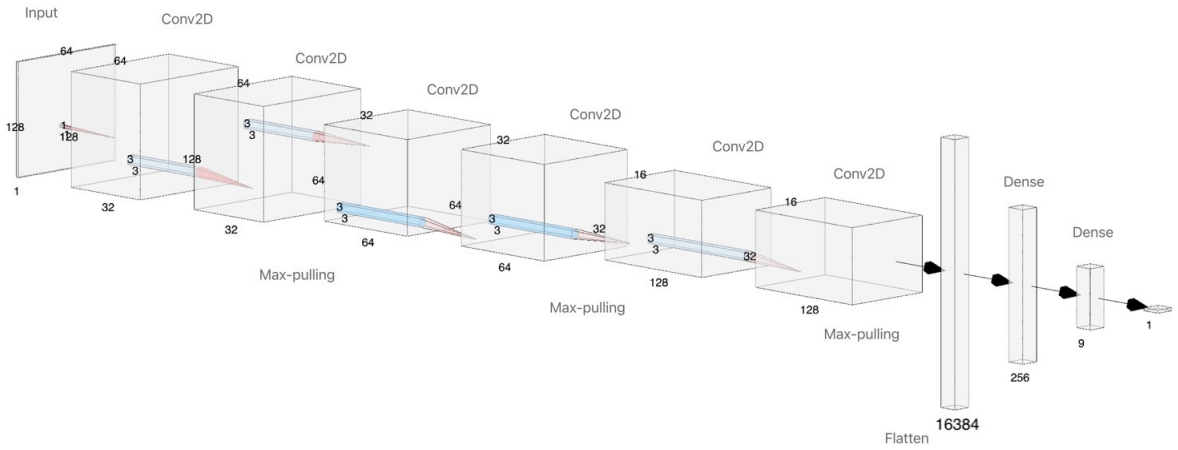

Figure 1: Inspired CNN Architecture

This model is inspired by the famous VGG16 model. We simplified the VGG16 model, and keeping developing based on this new model.

3.2 Simple model

Algorithm 1 Simple Model

1: model call Sequential()
2: model add(ZeroPadding2D((1,1),input_shape=(64,128,1)))
3: Model call Conv2D()with 3x3x10 filters,activate with relu
4: Model call MaxPooling(2,2)
5: Model call Conv2D()with 3x3x20 filters,activate with relu
6: Model call MaxPooling(2,2)
7: Model call Flatten()convert to 1d
8: Model minimize output by call Dense(), activate with relu
9: Model add(Dropout(0.5))
10: Model add(Dense(9, activation='softmax'))

We also add another hidden layer after max pooling to 8x16x128, we convolution the data to 4x8x200, to see how deep the layer would influence our result.

And a simple hidden layer with only two convulsions, one of it with 10 fillers and another with 20 filters

3.3 Backward propagation

. We use build-in method model.compile() in Keras, it will do backward propagation for us. We only have to define the loss function and optimizer that we want use. In this experiment we use cross entropy as the loss function and Adam as our optimizer.

3.4 Decisions and Selections

. Although most of the detail about building our network was already discussed in the forward propagation section there are other important decisions and findings.

3.4.1 Validation set

Kera is extremely powerful and convenient to customize the training process. A validation rate = 0.33, that is, 66% of the training set were kept, and the rest of the training set will automatically be set to be a validation set and do the validation test. It will help the result to be more stable, and the validation accuracy is an important factor that decides whether we should submit the result to Kaggle or not.

3.4.2 Drop out rate

At the early models that were built, a drop rate of 0.5 is used which is the same as the original VGG16 model. However we found that the validation accuracy is about 5% to 10% higher than the training accuracy after we apply validation set separation on our models. So we changed 0.5 to 0.3 and the validation accuracy and training accuracy became very close.

4 Results

4.1 Bernoulli Naïve Bayes

	Trainning Accuracy	Validation Accuracy
simple model	0.9799	0.1902
VGG base model 8 layers	0.8091	0.8417
VGG 10 layers	0.7464	0.8270
VGG 10 layers & dropout = 0.3 & earlystopping	0.8632	0.8701

Table 2: Train accuray and Validation accuary Table for different Neural Network models

By using 6 K-fold to training and validation the data, and the final average of them. As the output shows, we got the accuracy with the train set and test set. The accuracy of the training set similar to the test set, which means the prediction result is reliable. There is also no big difference between the accuracy of each fold. The accuracy is influenced by the choice of words, so with the method, we used for choosing words, the final average accuracy is about 80 percent.

5 Discussion and Conclusion

We found the deeper hidden layer would directly influence the result. At the beginning of the experiment, we just used convolution twice with small number filters and did not include any zero-padding. The result only achieves around 18.9% percent of the correct picture. We

also try a complex neural network with a deep layer, it takes a long time to train the data and the improvement of prediction is not remarkable. In other words, it is not necessary to build a neural network with too many layers in our experiment.

We also found choosing a filter as much as you can obviously improve the accuracy since it includes more information about the picture. Except that, the most important thing is you have to call the dropout function to avoid getting too much noise and information since they would result in the training overfitting.

In addition, choosing enough of the batch size is necessary, since it keeps the accuracy of the gradient descent. If the batch size is too small the gradient descent would be influenced, if it is too large it would take too much of the memory. We finally choose a batch size of 100, it makes more sense for our experiment. Also training more time would be also important, when we set the epoch to 4, we realize the iteration is not enough to get an accurate prediction until we switch to epoch to 20 and the result achieves sufficiently good output.

From other researches, we know that ensemble learning will improve the final accuracy, and it is our direction for future investigation.

6 Statement of Contributions

Peng Li: Implemented simply cnn, and try different hidden layers of the cnn. Also find what could affect the experiment accuracy.
Chengyi Song: Modification on the test part of the provided code. Try to use the torch for this project.
Zeye Gu: Implemented the inspired versions of the VGG networks, tested the accuracy with different hyper-parameters. Processed the report by using LaTex.

REFERENCE

Neuhive(2018, November 20) VGG16 – Convolutional Network for Classification and Detection. Retrieved December 06, 2020 from https://neurohive.io/en/popular-networks/vgg16/

ashishpatel26 (2020,September 2) Tools to Design or Visualize Architecture of Neural Network. Retrieved December 06, 2020, from https://github.com/ashishpatel26/Tools-to-Design-or-Visualize-Architecture-of-Neural-Network

Rohit Thakur(2019,Aug 6) Step by step VGG16 implementation in Keras for beginners Retrieved December 06, 2020, from https://towardsdatascience.com/step-by-step-vgg16-implementation-in-keras-for-beginners-a833c686ae6c

Neuhive(2018, November 20) VGG16 – Convolutional Network for Classification and Detection. Retrieved December 06, 2020 from https://neurohive.io/en/popular-networks/vgg16/

ashishpatel26 (2020,September 2) Tools to Design or Visualize Architecture of Neural Network. Retrieved December 06, 2020, from https://github.com/ashishpatel26/Tools-to-Design-or-Visualize-Architecture-of-Neural-Network

```
from google.colab import drive
drive.mount('/content/gdrive' )
```

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

```
%cd '/content/gdrive/My Drive/COMP4900/a3/'
!ls './data/'
```

```
/content/gdrive/My Drive/COMP4900/a3
Test.pkl  TrainLabels.csv  Train.pkl
```

```
# basic
import io
import cv2
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D, LeakyReLU, Ze
from keras.utils import plot_model, to_categorical
from keras.callbacks import History
from keras.optimizers import Nadam, Adam, SGD
import matplotlib.pyplot as plt
from keras.utils import np_utils
from keras import backend as K
from keras import regularizers

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import OneHotEncoder
```

```
train_labels = pd.read_csv('./data/TrainLabels.csv')
train_images = pd.read_pickle('./data/Train.pkl')
test_images = pd.read_pickle('./data/Test.pkl')

# train_images = np.where(train_images<250, 0, train_images)
# test_images = np.where(test_images<250, 0, test_images)

X_train = np.array(train_images)
X_train = np.array(X_train).reshape(-1,64,128,1)
onehot_encoder = OneHotEncoder(sparse=False)
```

```
onehot_encoder = OneHotEncoder(sparse=False)
y_train = onehot_encoder.fit_transform(np.reshape(np.array(train_labels['class']), (-1

X_test = np.array(test_images)
X_test = np.array(X_test).reshape(-1,64,128,1)


print("X_train.shape:",X_train.shape)
print("X_test.shape:",X_test.shape)
print("y_train.shape:",y_train.shape)
```

```
X_train.shape: (60000, 64, 128, 1)
X_test.shape: (10000, 64, 128, 1)
y_train.shape: (60000, 9)
```

base model 8 layer (maxPolling layers aren't counted here)

```
baseModel = Sequential([
                        ZeroPadding2D((1,1),input_shape=(64,128,1)),
                        Conv2D(32, kernel_size=(3, 3), activation='relu'),
                        ZeroPadding2D((1,1)),
                        Conv2D(32, kernel_size=(3, 3), activation='relu'),
                        MaxPooling2D(pool_size=(2, 2)),

                        ZeroPadding2D((1,1)),
                        Conv2D(64, kernel_size=(3, 3), activation='relu'),
                        ZeroPadding2D((1,1)),
                        Conv2D(64, kernel_size=(3, 3), activation='relu'),
                        MaxPooling2D(pool_size=(2, 2)),
                        Dropout(0.5),

                        ZeroPadding2D((1,1)),
                        Conv2D(128, kernel_size=(3, 3), activation='relu'),
                        ZeroPadding2D((1,1)),
                        Conv2D(128, kernel_size=(3, 3), activation='relu'),
                        MaxPooling2D(pool_size=(2, 2)),
                        Dropout(0.5),

                        Flatten(),
                        Dense(512, activation='relu'),
                        Dropout(0.5),
                        Dense(9, activation='softmax'),

])

baseModel.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accurac
```

```
baseModel.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
zero_padding2d (ZeroPadding2 (None, 66, 130, 1)        0
_________________________________________________________________
conv2d (Conv2D)              (None, 64, 128, 32)       320
_________________________________________________________________
zero_padding2d_1 (ZeroPaddin (None, 66, 130, 32)       0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 64, 128, 32)       9248
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 32, 64, 32)        0
_________________________________________________________________
zero_padding2d_2 (ZeroPaddin (None, 34, 66, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 32, 64, 64)        18496
_________________________________________________________________
zero_padding2d_3 (ZeroPaddin (None, 34, 66, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 32, 64, 64)        36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 32, 64)        0
_________________________________________________________________
dropout (Dropout)            (None, 16, 32, 64)        0
_________________________________________________________________
zero_padding2d_4 (ZeroPaddin (None, 18, 34, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 16, 32, 128)       73856
_________________________________________________________________
zero_padding2d_5 (ZeroPaddin (None, 18, 34, 128)       0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 16, 32, 128)       147584
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 8, 16, 128)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 8, 16, 128)        0
_________________________________________________________________
flatten (Flatten)            (None, 16384)             0
_________________________________________________________________
dense (Dense)                (None, 512)               8389120
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 9)                 4617
=================================================================
Total params: 8,680,169
Trainable params: 8,680,169
Non-trainable params: 0
_________________________________________________________________
```

```
history = baseModel.fit(X_train, y_train, batch_size=100, epochs=20, validation_split

print('Finished Training')
```

```
predictions = np.argmax(baseModel.predict(X_test), axis=1)
```

```
Epoch 1/20
402/402 [==============================] - 10s 26ms/step - loss: 2.2938 - accura
Epoch 2/20
402/402 [==============================] - 10s 25ms/step - loss: 2.0906 - accura
Epoch 3/20
402/402 [==============================] - 10s 25ms/step - loss: 1.7752 - accura
Epoch 4/20
402/402 [==============================] - 10s 25ms/step - loss: 1.5694 - accura
Epoch 5/20
402/402 [==============================] - 10s 25ms/step - loss: 1.4611 - accura
Epoch 6/20
402/402 [==============================] - 10s 25ms/step - loss: 1.3815 - accura
Epoch 7/20
402/402 [==============================] - 10s 25ms/step - loss: 1.3174 - accura
Epoch 8/20
402/402 [==============================] - 10s 25ms/step - loss: 1.2603 - accura
Epoch 9/20
402/402 [==============================] - 10s 25ms/step - loss: 1.2010 - accura
Epoch 10/20
402/402 [==============================] - 10s 25ms/step - loss: 1.1479 - accura
Epoch 11/20
402/402 [==============================] - 10s 25ms/step - loss: 1.0660 - accura
Epoch 12/20
402/402 [==============================] - 10s 25ms/step - loss: 0.9827 - accura
Epoch 13/20
402/402 [==============================] - 10s 25ms/step - loss: 0.9004 - accura
Epoch 14/20
402/402 [==============================] - 10s 25ms/step - loss: 0.8083 - accura
Epoch 15/20
402/402 [==============================] - 10s 25ms/step - loss: 0.7612 - accura
Epoch 16/20
402/402 [==============================] - 10s 25ms/step - loss: 0.7064 - accura
Epoch 17/20
402/402 [==============================] - 10s 25ms/step - loss: 0.6681 - accura
Epoch 18/20
402/402 [==============================] - 10s 25ms/step - loss: 0.6333 - accura
Epoch 19/20
402/402 [==============================] - 10s 25ms/step - loss: 0.6144 - accura
Epoch 20/20
402/402 [==============================] - 10s 25ms/step - loss: 0.5879 - accura
Finished Training
```

10 Layers & dropreate =0.5

```
model2 = Sequential([
                        ZeroPadding2D((1,1),input_shape=(64,128,1)),
                        Conv2D(32, kernel_size=(3, 3), activation='relu'),
                        ZeroPadding2D((1,1)),
                        Conv2D(32, kernel_size=(3, 3), activation='relu'),
                        MaxPooling2D(pool_size=(2, 2)),
```

```
                        ZeroPadding2D((1,1)),
                        Conv2D(64, kernel_size=(3, 3), activation='relu'),
                        ZeroPadding2D((1,1)),
                        Conv2D(64, kernel_size=(3, 3), activation='relu'),
                        MaxPooling2D(pool_size=(2, 2)),
                        Dropout(0.5),

                        ZeroPadding2D((1,1)),
                        Conv2D(128, kernel_size=(3, 3), activation='relu'),
                        ZeroPadding2D((1,1)),
                        Conv2D(128, kernel_size=(3, 3), activation='relu'),
                        MaxPooling2D(pool_size=(2, 2)),
                        Dropout(0.5),

                        ZeroPadding2D((1,1)),
                        Conv2D(256, kernel_size=(3, 3), activation='relu'),
                        ZeroPadding2D((1,1)),
                        Conv2D(256, kernel_size=(3, 3), activation='relu'),
                        MaxPooling2D(pool_size=(2, 2)),
                        Dropout(0.5),

                        Flatten(),
                        Dense(512, activation='relu'),
                        Dropout(0.5),
                        Dense(9, activation='softmax'),

])

model2.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy']

model2.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
zero_padding2d_6 (ZeroPaddin (None, 66, 130, 1)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 64, 128, 32)       320
_________________________________________________________________
zero_padding2d_7 (ZeroPaddin (None, 66, 130, 32)       0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 64, 128, 32)       9248
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 32, 64, 32)        0
_________________________________________________________________
zero_padding2d_8 (ZeroPaddin (None, 34, 66, 32)        0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 32, 64, 64)        18496
_________________________________________________________________
zero_padding2d_9 (ZeroPaddin (None, 34, 66, 64)        0
_________________________________________________________________
```

```
conv2d_9 (Conv2D)            (None, 32, 64, 64)        36928
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 16, 32, 64)        0
_________________________________________________________________
dropout_3 (Dropout)          (None, 16, 32, 64)        0
_________________________________________________________________
zero_padding2d_10 (ZeroPaddi (None, 18, 34, 64)        0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 16, 32, 128)       73856
_________________________________________________________________
zero_padding2d_11 (ZeroPaddi (None, 18, 34, 128)       0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 16, 32, 128)       147584
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 8, 16, 128)        0
_________________________________________________________________
dropout_4 (Dropout)          (None, 8, 16, 128)        0
_________________________________________________________________
zero_padding2d_12 (ZeroPaddi (None, 10, 18, 128)       0
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 8, 16, 256)        295168
_________________________________________________________________
zero_padding2d_13 (ZeroPaddi (None, 10, 18, 256)       0
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 8, 16, 256)        590080
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 4, 8, 256)         0
_________________________________________________________________
dropout_5 (Dropout)          (None, 4, 8, 256)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               4194816
_________________________________________________________________
dropout_6 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 9)                 4617
=================================================================
Total params: 5,371,113
```

```
history = model2.fit(X_train, y_train, batch_size=100, epochs=20, validation_split = (
```

```
print('Finished Training')
```

```
predictions2 = np.argmax(model2.predict(X_test), axis=1)
```

```
Epoch 1/20
  1/402 [..............................] - ETA: 0s - loss: 8.9604 - accuracy: 0.(
402/402 [==============================] - 21s 52ms/step - loss: 2.2614 - accura(
Epoch 2/20
402/402 [==============================] - 20s 50ms/step - loss: 2.1975 - accura(
Epoch 3/20
402/402 [==============================] - 20s 50ms/step - loss: 2.1975 - accura(
Epoch 4/20
402/402 [==============================] - 20s 50ms/step - loss: 2.1974 - accura(
Epoch 5/20
```

```
402/402 [==============================] - 20s 50ms/step - loss: 2.1977 - accura
Epoch 6/20
402/402 [==============================] - 20s 50ms/step - loss: 2.1972 - accura
Epoch 7/20
402/402 [==============================] - 20s 50ms/step - loss: 1.9058 - accura
Epoch 8/20
402/402 [==============================] - 20s 50ms/step - loss: 1.6229 - accura
Epoch 9/20
402/402 [==============================] - 20s 50ms/step - loss: 1.5148 - accura
Epoch 10/20
402/402 [==============================] - 20s 50ms/step - loss: 1.4414 - accura
Epoch 11/20
402/402 [==============================] - 20s 50ms/step - loss: 1.3761 - accura
Epoch 12/20
402/402 [==============================] - 20s 50ms/step - loss: 1.3285 - accura
Epoch 13/20
402/402 [==============================] - 20s 50ms/step - loss: 1.2848 - accura
Epoch 14/20
402/402 [==============================] - 20s 50ms/step - loss: 1.2372 - accura
Epoch 15/20
402/402 [==============================] - 20s 50ms/step - loss: 1.2093 - accura
Epoch 16/20
402/402 [==============================] - 20s 50ms/step - loss: 1.1594 - accura
Epoch 17/20
402/402 [==============================] - 20s 50ms/step - loss: 1.0885 - accura
Epoch 18/20
402/402 [==============================] - 20s 50ms/step - loss: 0.9967 - accura
Epoch 19/20
402/402 [==============================] - 20s 50ms/step - loss: 0.8974 - accura
Epoch 20/20
402/402 [==============================] - 20s 50ms/step - loss: 0.8128 - accura
Finished Training
```

10 Layers & dropreate =0.3 plus earlystopping

```
model3 = Sequential([
                        ZeroPadding2D((1,1),input_shape=(64,128,1)),
                        Conv2D(32, kernel_size=(3, 3), activation='relu'),
                        ZeroPadding2D((1,1)),
                        Conv2D(32, kernel_size=(3, 3), activation='relu'),
                        MaxPooling2D(pool_size=(2, 2)),

                        ZeroPadding2D((1,1)),
                        Conv2D(64, kernel_size=(3, 3), activation='relu'),
                        ZeroPadding2D((1,1)),
                        Conv2D(64, kernel_size=(3, 3), activation='relu'),
                        MaxPooling2D(pool_size=(2, 2)),
                        Dropout(0.3),

                        ZeroPadding2D((1,1)),
                        Conv2D(128, kernel_size=(3, 3), activation='relu'),
                        ZeroPadding2D((1,1)),
```

```
                        Conv2D(128, kernel_size=(3, 3), activation='relu'),
                        MaxPooling2D(pool_size=(2, 2)),
                        Dropout(0.3),

                        ZeroPadding2D((1,1)),
                        Conv2D(256, kernel_size=(3, 3), activation='relu'),
                        ZeroPadding2D((1,1)),
                        Conv2D(256, kernel_size=(3, 3), activation='relu'),
                        MaxPooling2D(pool_size=(2, 2)),
                        Dropout(0.3),

                        Flatten(),
                        Dense(512, activation='relu'),
                        Dropout(0.3),
                        Dense(9, activation='softmax'),

    ])

    model3.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy']

    model3.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
zero_padding2d (ZeroPadding2 (None, 66, 130, 1)        0
_________________________________________________________________
conv2d (Conv2D)              (None, 64, 128, 32)       320
_________________________________________________________________
zero_padding2d_1 (ZeroPaddin (None, 66, 130, 32)       0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 64, 128, 32)       9248
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 32, 64, 32)        0
_________________________________________________________________
zero_padding2d_2 (ZeroPaddin (None, 34, 66, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 32, 64, 64)        18496
_________________________________________________________________
zero_padding2d_3 (ZeroPaddin (None, 34, 66, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 32, 64, 64)        36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 32, 64)        0
_________________________________________________________________
dropout (Dropout)            (None, 16, 32, 64)        0
_________________________________________________________________
zero_padding2d_4 (ZeroPaddin (None, 18, 34, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 16, 32, 128)       73856
_________________________________________________________________
zero_padding2d_5 (ZeroPaddin (None, 18, 34, 128)       0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 16, 32, 128)       147584
```

```
max_pooling2d_2 (MaxPooling2 (None, 8, 16, 128)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 8, 16, 128)        0
_________________________________________________________________
zero_padding2d_6 (ZeroPaddin (None, 10, 18, 128)       0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 8, 16, 256)        295168
_________________________________________________________________
zero_padding2d_7 (ZeroPaddin (None, 10, 18, 256)       0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 8, 16, 256)        590080
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 4, 8, 256)         0
_________________________________________________________________
dropout_2 (Dropout)          (None, 4, 8, 256)         0
_________________________________________________________________
flatten (Flatten)            (None, 8192)              0
_________________________________________________________________
dense (Dense)                (None, 512)               4194816
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 9)                 4617
=================================================================
Total params: 5,371,113
```

```
from tensorflow.keras.callbacks import EarlyStopping

callback = EarlyStopping(monitor='val_loss', patience=3, mode="auto",)
history = model3.fit(X_train, y_train, batch_size=100, epochs=20, validation_split = (


print('Finished Training')

predictions3 = np.argmax(model3.predict(X_test), axis=1)
```

```
Epoch 1/20
402/402 [==============================] - 12s 30ms/step - loss: 2.2378 - accura
Epoch 2/20
402/402 [==============================] - 11s 28ms/step - loss: 2.1974 - accura
Epoch 3/20
402/402 [==============================] - 11s 28ms/step - loss: 2.1976 - accura
Epoch 4/20
402/402 [==============================] - 11s 28ms/step - loss: 2.1974 - accura
Epoch 5/20
402/402 [==============================] - 11s 28ms/step - loss: 1.9575 - accura
Epoch 6/20
402/402 [==============================] - 11s 28ms/step - loss: 1.5455 - accura
Epoch 7/20
402/402 [==============================] - 11s 28ms/step - loss: 1.3949 - accura
Epoch 8/20
402/402 [==============================] - 11s 28ms/step - loss: 1.2790 - accura
```

```
Epoch 9/20
402/402 [==============================] - 11s 28ms/step - loss: 1.1815 - accura
Epoch 10/20
402/402 [==============================] - 12s 29ms/step - loss: 1.0448 - accura
Epoch 11/20
402/402 [==============================] - 11s 28ms/step - loss: 0.8733 - accura
Epoch 12/20
402/402 [==============================] - 11s 28ms/step - loss: 0.7316 - accura
Epoch 13/20
402/402 [==============================] - 11s 28ms/step - loss: 0.6525 - accura
Epoch 14/20
402/402 [==============================] - 11s 28ms/step - loss: 0.5935 - accura
Epoch 15/20
402/402 [==============================] - 11s 28ms/step - loss: 0.5626 - accura
Epoch 16/20
402/402 [==============================] - 11s 28ms/step - loss: 0.5382 - accura
Epoch 17/20
402/402 [==============================] - 11s 28ms/step - loss: 0.5083 - accura
Epoch 18/20
402/402 [==============================] - 11s 28ms/step - loss: 0.4667 - accura
Epoch 19/20
402/402 [==============================] - 11s 28ms/step - loss: 0.4556 - accura
Epoch 20/20
402/402 [==============================] - 11s 28ms/step - loss: 0.4357 - accura
Finished Training
```

Simple model

```
# model.add(Conv2D(10, kernel_size=(3, 3), activation='relu'))
#     model.add(MaxPooling2D(pool_size=(2, 2)))
#     model.add(Conv2D(20, kernel_size=(3, 3), activation='relu'))
#     model.add(MaxPooling2D(pool_size=(2, 2)))
#     model.add(Flatten())
#     model.add(Dense(512, activation='relu'))
#     model.add(Dense(9, activation='softmax'))


simpleModel = Sequential([
                        ZeroPadding2D((1,1),input_shape=(64,128,1)),
                        Conv2D(10, kernel_size=(3, 3), activation='relu'),
                        MaxPooling2D(pool_size=(2, 2)),
                        Conv2D(20, kernel_size=(3, 3), activation='relu'),
                        MaxPooling2D(pool_size=(2, 2)),


                        Flatten(),
                        Dense(512, activation='relu'),
                        Dense(9, activation='softmax'),

])
```

```
simpleModel.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accu

simpleModel.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
zero_padding2d_14 (ZeroPaddi (None, 66, 130, 1)        0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 64, 128, 10)       100
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 32, 64, 10)        0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 30, 62, 20)        1820
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 15, 31, 20)        0
_________________________________________________________________
flatten_2 (Flatten)          (None, 9300)              0
_________________________________________________________________
dense_4 (Dense)              (None, 512)               4762112
_________________________________________________________________
dense_5 (Dense)              (None, 9)                 4617
=================================================================
Total params: 4,768,649
Trainable params: 4,768,649
Non-trainable params: 0
_________________________________________________________________
```

```
history_Simple = simpleModel.fit(X_train, y_train, batch_size=100, epochs=20, validati

print('Finished Training')

predictions_Simple = np.argmax(simpleModel.predict(X_test), axis=1)
```

```
Epoch 1/20
402/402 [==============================] - 4s 9ms/step - loss: 3.2128 - accuracy
Epoch 2/20
402/402 [==============================] - 4s 9ms/step - loss: 2.1638 - accuracy
Epoch 3/20
402/402 [==============================] - 4s 9ms/step - loss: 1.9770 - accuracy
Epoch 4/20
402/402 [==============================] - 3s 9ms/step - loss: 1.7288 - accuracy
Epoch 5/20
402/402 [==============================] - 4s 9ms/step - loss: 1.4465 - accuracy
Epoch 6/20
402/402 [==============================] - 3s 9ms/step - loss: 1.0618 - accuracy
Epoch 7/20
402/402 [==============================] - 3s 9ms/step - loss: 0.6807 - accuracy
Epoch 8/20
402/402 [==============================] - 3s 9ms/step - loss: 0.3909 - accuracy
Epoch 9/20
402/402 [==============================] - 3s 9ms/step - loss: 0.2307 - accuracy
Epoch 10/20
402/402 [==============================] - 3s 9ms/step - loss: 0.1740 - accuracy
Epoch 11/20
```

```
402/402 [==============================] - 3s 9ms/step - loss: 0.1354 - accuracy
Epoch 12/20
402/402 [==============================] - 3s 9ms/step - loss: 0.1218 - accuracy
Epoch 13/20
402/402 [==============================] - 3s 9ms/step - loss: 0.1227 - accuracy
Epoch 14/20
402/402 [==============================] - 3s 9ms/step - loss: 0.1029 - accuracy
Epoch 15/20
402/402 [==============================] - 3s 9ms/step - loss: 0.0694 - accuracy
Epoch 16/20
402/402 [==============================] - 3s 9ms/step - loss: 0.0645 - accuracy
Epoch 17/20
402/402 [==============================] - 4s 9ms/step - loss: 0.0692 - accuracy
Epoch 18/20
402/402 [==============================] - 3s 9ms/step - loss: 0.0824 - accuracy
Epoch 19/20
402/402 [==============================] - 3s 9ms/step - loss: 0.0810 - accuracy
Epoch 20/20
402/402 [==============================] - 4s 9ms/step - loss: 0.0640 - accuracy
Finished Training
```

```
results = np.array(list(zip(np.arange(0,10000),predictions3+5)))
results = pd.DataFrame(results, columns=['id', 'class'])
results.to_csv('submit34.csv', index = False)
```