

Character Recognition

Jingduo
Guo

Yang
Luo

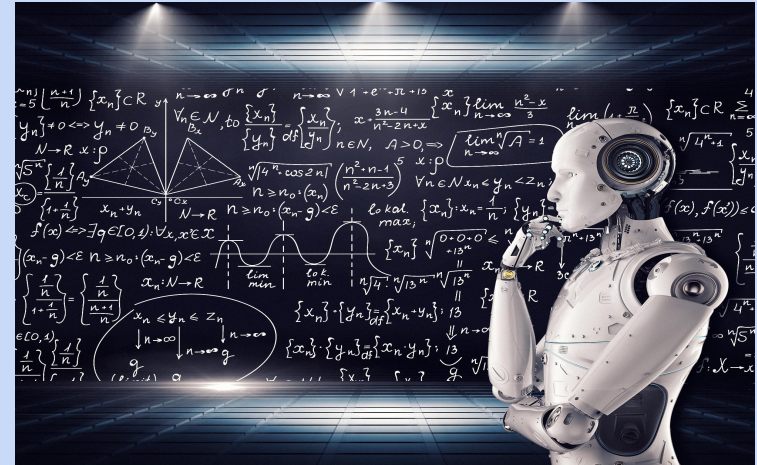
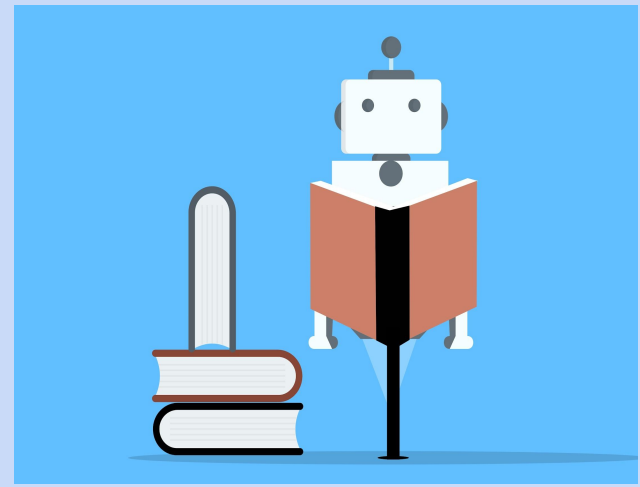
Wendy
Zhai

Honghui
LI



Introduction

A popular demonstration of the capability of deep learning techniques is object recognition in image data. This deep learning application in python recognizes alphabet through gestures captured real-time on a webcam. The user is allowed to write the alphabet on the screen using an object-of-interest



Character Recognition Using ML

What we will do?

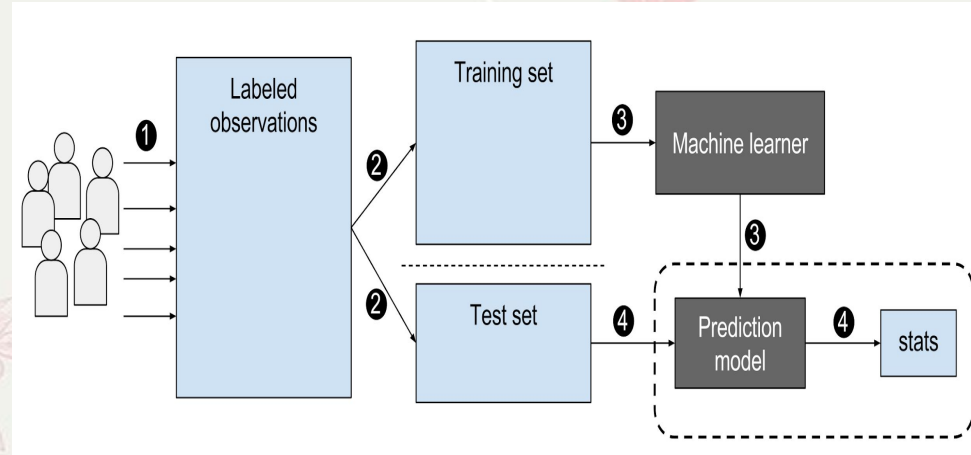
We will take data out which we will be separating them into two parts:

1 Test data

2 Training data

We will train our algorithm through test data and training data to use for our ANN model. So our model is trained for letter recognition

After that we will use it for predicting purpose and calculate the accuracy of our model



Character Recognition Using ML

What we will do?

The code

We basically use sklearn library to import and split our train data:

```
ImageDatas = []
files = dataset["image"]
label = dataset["label"]
i = 0
print("====starting====")
for fileName in files:

    image=tf.keras.utils.load_img(os.path.join(directory,fileName),color_
    mode='grayscale',target_size=(100,100))
    image=tf.keras.utils.img_to_array(image)
    image=image/255.0
    imageDatas+=[image]
    i = i + 1
    if (i % 500 == 0):
        print(i)
```

```
class_names = dataset["label"].unique()
print(class_names)
print("====done====")
```

```
====starting====
500
1000
1500
2000
2500
3000
['0' '1' '2' '3' '4' '5' '6' '7' '8' '9' 'A' 'B' 'C' 'D' 'E' 'F' 'G' 'H'
 'I' 'J' 'K' 'L' 'M' 'N' 'O' 'P' 'Q' 'R' 'S' 'T' 'U' 'V' 'W' 'X' 'Y' 'Z'
 'a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j' 'k' 'l' 'm' 'n' 'o' 'p' 'q' 'r'
 's' 't' 'u' 'v' 'w' 'x' 'y' 'z']
====done====
```


Character Recognition Using ML

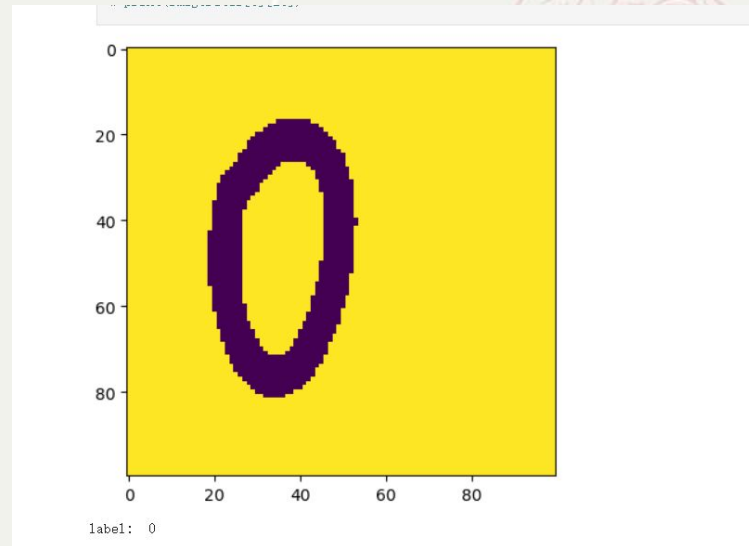
What we will do?

The code...

Then we verify the image data and print label imagedata result.

```
directory = "archive"  
showingImage=tf.keras.utils.load_img(os.path.join(directory,files[0])  
,color_mode='grayscale',target_size=(100,100))  
plt.imshow(imageDatas[0])  
plt.show()
```

```
print("label: ", label[0])  
# print(imageDatas[0][20])
```



Character Recognition Using ML

What we will do?

The code...

```
ImageData_compressed = []

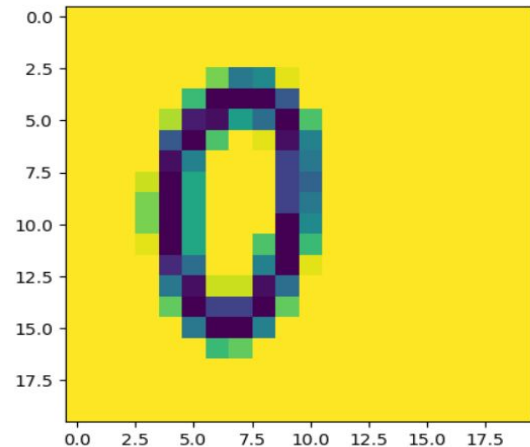
print(np.array(imageDatas[0]).shape)
# test = []
# test += [imageDatas[0]]
count = 0
print("====start compressing====")
for img in imageDatas:
    img_compressed = []
    for i in range(20):
        row_compressed = []
        for j in range(20):
            sum = 0
            for r in range(5):
                for c in range(5):
                    sum += img[5 * i + r][5 * j + c]
            row_compressed += [sum/25]
        #
    print(np.array(row_compressed).shape)
```

Then we compressed file to 20x20 version

```
img_compressed +=
[row_compressed]
#
print(np.array(img_compressed).shape)
ImageData_compressed +=
[img_compressed]
count += 1
if (count % 500 == 0):
    print("compressed: ", count, " imgs ")

print("====done compressing====")
print(np.array(ImageData_compressed).shape)
plt.imshow(ImageData_compressed[0])
# print(ImageData_compressed[0])
plt.show()
```

```
(100, 100, 1)
compressed: 500 imgs
compressed: 1000 imgs
compressed: 1500 imgs
compressed: 2000 imgs
compressed: 2500 imgs
compressed: 3000 imgs
==
(3410, 20, 20, 1)
```



Character Recognition Using ML

What we will do?

The code...

Then we process label to find match number

```
def getIndex(letter):  
    index = ord(letter)-48  
    if (index > 10):  
        index -= 7  
    if (index > 35):  
        index -= 6  
    return index
```

```
# process label to number  
letters = label.unique()  
print(letters)
```

```
# checking  
indexes = []  
for l in letters:  
    indexes += [getIndex(l)]
```

```
print(indexs)  
for i in range(61):  
    if (i != indexs[i]):  
        print("error in: ", i)
```

```
['0' '1' '2' '3' '4' '5' '6' '7' '8' '9' 'A' 'B' 'C' 'D' 'E' 'F' 'G' 'H'  
'I' 'J' 'K' 'L' 'M' 'N' 'O' 'P' 'Q' 'R' 'S' 'T' 'U' 'V' 'W' 'X' 'Y' 'Z'  
'a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j' 'k' 'l' 'm' 'n' 'o' 'p' 'q' 'r'  
's' 't' 'u' 'v' 'w' 'x' 'y' 'z']
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61]
```

Character Recognition Using ML

What we will do?

The code...

we split data

```
data_nparr=np.array(ImageData_compressed)
```

```
#
```

```
data_nparr=np.array(ImageData_compressed_type2)
```

```
# data_nparr=np.array(imageDatas)
```

```
label_nparr=np.array(label)
```

```
label_index_nparr =
```

```
np.vectorize(getIndex)(label_nparr)
```

```
train_data, test_data, train_label,
```

```
test_label = train_test_split(data_nparr,
```

```
label_index_nparr, test_size=0.1,
```

```
random_state=42)
```

```
print("data splited")
```

Then tensorflow library to import **datasets**,
layers, **models**

```
import tensorflow as tf
```

```
from tensorflow.keras import datasets,  
layers, models
```

```
model = models.Sequential()
```

```
model.add(layers.Conv2D(32, (3, 3),  
activation='relu', input_shape=(20, 20, 1)))
```

```
# compressed (20 x 20)
```

```
# model.add(layers.Conv2D(32, (3, 3),  
activation='relu', input_shape=(100, 100,
```

```
1))) # uncompressed (100 x 100)
```

```
model.add(layers.MaxPooling2D((2, 2)))
```

```
model.add(layers.Conv2D(64, (3, 3),  
activation='relu'))
```

```
model.add(layers.MaxPooling2D((2, 2)))
```

```
model.add(layers.Conv2D(64, (3, 3),  
activation='relu'))
```

```
layer added  
Model: "sequential_3"
```

Layer (type)	Output Shape	Param #
conv2d_12 (Conv2D)	(None, 18, 18, 32)	320
max_pooling2d_6 (MaxPooling2D)	(None, 9, 9, 32)	0
conv2d_13 (Conv2D)	(None, 7, 7, 64)	18496
max_pooling2d_7 (MaxPooling2D)	(None, 3, 3, 64)	0
conv2d_14 (Conv2D)	(None, 1, 1, 64)	36928
flatten_3 (Flatten)	(None, 64)	0
dense_6 (Dense)	(None, 64)	4160
dense_7 (Dense)	(None, 62)	4030
=====		
Total params: 63,934		
Trainable params: 63,934		
Non-trainable params: 0		

Character Recognition Using ML

What we will do?

The code...

```
model.add(layers.MaxPooling2D((2, 2)))  
model.add(layers.Conv2D(64, (3, 3),  
activation='relu'))
```

```
model.add(layers.Flatten())  
model.add(layers.Dense(64,  
activation='relu'))  
model.add(layers.Dense(62,  
activation='softmax'))  
print("layer added")  
model.summary()
```

```
layer added  
Model: "sequential_3"
```

Layer (type)	Output Shape	Param #
=====		
conv2d_12 (Conv2D)	(None, 18, 18, 32)	320
max_pooling2d_6 (MaxPooling 2D)	(None, 9, 9, 32)	0
conv2d_13 (Conv2D)	(None, 7, 7, 64)	18496
max_pooling2d_7 (MaxPooling 2D)	(None, 3, 3, 64)	0
conv2d_14 (Conv2D)	(None, 1, 1, 64)	36928
flatten_3 (Flatten)	(None, 64)	0
dense_6 (Dense)	(None, 64)	4160
dense_7 (Dense)	(None, 62)	4030
=====		
Total params: 63,934		
Trainable params: 63,934		
Non-trainable params: 0		

Character Recognition Using ML

What we will do?

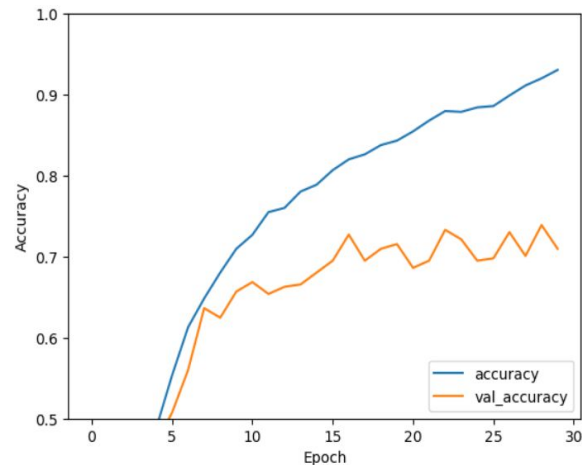
The code...

```
model.compile(optimizer='adam',  
loss=tf.keras.losses.SparseCategoricalC  
rossentropy(from_logits=False),  
metrics=['accuracy'])  
  
print("====start training====")  
history = model.fit(train_data,  
train_label, epochs=30,  
validation_data=(test_data,  
test_label))  
print("====done training====")
```

```
plt.plot(history.history['accuracy'],  
label='accuracy')  
plt.plot(history.history['val_accuracy'],  
label = 'val_accuracy')  
plt.xlabel('Epoch')  
plt.ylabel('Accuracy')  
plt.ylim([0.5, 1])  
plt.legend(loc='lower right')
```

```
test_loss, test_acc =  
model.evaluate(test_data, test_label,  
verbose=2)
```

11/11 - 0s - loss: 1.1248 - accuracy: 0.7097 - 37ms/epoch - 3ms/step



compressed vs. uncompressed performance difference

Compressed

```
ImageData_compressed = []  
compressRatio = 4  
dimention = int(100 / compressRatio) #  
final dimention
```

```
print(np.array(imageDatas[0]).shape)
```

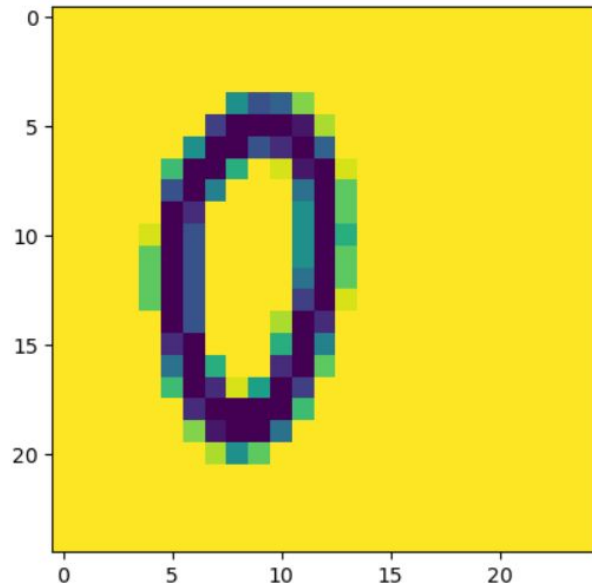
```
# test = []  
# test += [imageDatas[0]]  
count = 0  
print("====start compressing====")  
for img in imageDatas:  
    img_compressed = []  
    for i in range(dimention):  
        row_compressed = []  
        for j in range(dimention):  
            avg = 0  
            for r in range(compressRatio):  
                for c in range(compressRatio):  
                    avg += (img[compressRatio  
* i + r][compressRatio * j + c]) /  
compressRatio  
            row_compressed += [avg/25]  
            #
```

```
print(np.array(row_compressed).shape)
```

```
        img_compressed +=  
[row_compressed]  
        #  
print(np.array(img_compressed).shape)  
        ImageData_compressed +=  
[img_compressed]  
        count += 1  
        if (count % 500 == 0):  
            print("compressed: ", count, "  
imgs" )
```

```
ImageData_compressed =  
np.array(ImageData_compressed)  
print("====done compressing====")  
print(ImageData_compressed.shape)  
plt.imshow(ImageData_compressed[0])  
plt.show()
```

```
(100, 100, 1)  
====start compressing====  
compressed: 500 imgs  
compressed: 1000 imgs  
compressed: 1500 imgs  
compressed: 2000 imgs  
compressed: 2500 imgs  
compressed: 3000 imgs  
====done compressing====  
(3410, 25, 25, 1)
```



compressed vs. uncompressed performance difference

Compressed

model.compile(optimizer='adam',

loss=tf.keras.losses.SparseCategoricalCr
ossentropy(from_logits=False),
metrics=['accuracy'])

print("====start training====")
history = model.fit(train_data, train_label,
epochs=5,
validation_data=(test_data,
test_label))
print("====done training====")

```
print("====done training====")
```

```
====start training====
```

```
Epoch 1/5
```

```
96/96 [=====] - 6s 58ms/step - loss: 4.0567 - accuracy: 0.0303 - val_loss: 3.7247 - val_accuracy: 0.0850
```

```
Epoch 2/5
```

```
96/96 [=====] - 6s 59ms/step - loss: 2.7238 - accuracy: 0.3102 - val_loss: 2.3619 - val_accuracy: 0.3724
```

```
Epoch 3/5
```

```
96/96 [=====] - 5s 55ms/step - loss: 1.5509 - accuracy: 0.5780 - val_loss: 1.5328 - val_accuracy: 0.5865
```

```
Epoch 4/5
```

```
96/96 [=====] - 5s 55ms/step - loss: 0.9994 - accuracy: 0.7208 - val_loss: 1.2946 - val_accuracy: 0.6716
```

```
Epoch 5/5
```

```
96/96 [=====] - 5s 57ms/step - loss: 0.7206 - accuracy: 0.7830 - val_loss: 1.2047 - val_accuracy: 0.7038
```

```
====done training====
```


compressed vs. uncompressed performance difference

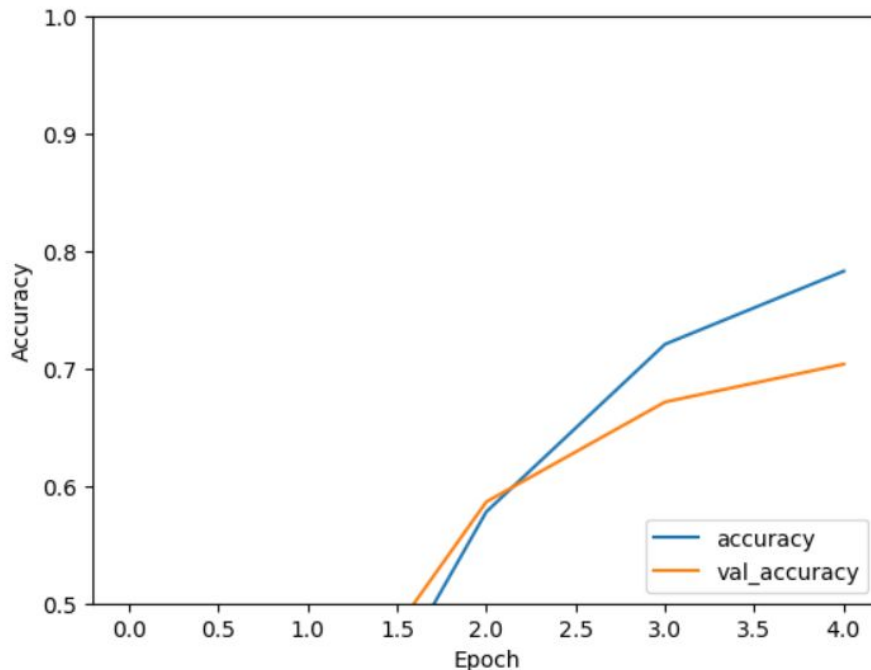
Compressed

```
import matplotlib.pyplot as plt  
plt.plot(history.history['accuracy'],  
label='accuracy')  
plt.plot(history.history['val_accuracy'],  
label='val_accuracy')  
plt.xlabel('Epoch')  
plt.ylabel('Accuracy')  
plt.ylim([0.5, 1])  
plt.legend(loc='lower right')
```

```
test_loss, test_acc =  
model.evaluate(test_data, test_label,  
verbose=2)
```

```
test_loss, test_acc = model.evaluate(test_data, test_label, verbose=2)
```

```
11/11 - 0s - loss: 1.2047 - accuracy: 0.7038 - 158ms/epoch - 14ms/step
```



compress vs. uncompressed performance difference

Uncompressed

```
import numpy
dimention = "100x100"
npzFileName = "Uncompressed_" +
dimention + ".npz"
```

```
imageDatas_np = np.array(imageDatas)
# Save compressed data to NPZ file
numpy.savez(npzFileName,
imageDatas_np=imageDatas_np)
print("=====done saving=====")
```

```
=====done saving=====
```

```
import numpy
import pandas as pd
import numpy as np
```

```
dataset = pd.read_csv('archive/english.csv')
directory = "archive"
```

```
imageDatas = []
files = dataset["image"]
label = dataset["label"]
dimention = "100x100"
npzFileName = "Uncompressed_" +
dimention + ".npz"
```

```
# Load compressed image matrix data
data = numpy.load(npzFileName)
ImageData_compressed =
data['imageDatas_np']
print("=====done load=====")
```

```
print(ImageData_compressed.shape)
```

```
from sklearn.model_selection import
train_test_split
```

```
In [1]: data_nparr=ImageData_compressed
#
data_nparr=np.array(ImageData_compre
ssed_type2)
# data_nparr=np.array(imageDatas)
label_nparr=np.array(label)
label_index_nparr =
np.vectorize(getIndex)(label_nparr)
```

```
train_data, test_data, train_label,
test_label =
train_test_split(data_nparr,
label_index_nparr, test_size=0.1,
random_state=42)
print("data splited")
```

```
import tensorflow as tf
```

```
from tensorflow.keras import
datasets, layers, models,
regularizers
```

```
print("Input shape:",
data_nparr[0].shape)
```

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3),
activation='relu',
input_shape=data_nparr[0].shape))
model.add(layers.MaxPooling2D((2,
2)))
```

compress vs. uncompressed performance difference

```
model.add(layers.Conv2D(64, (3, 3),
activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3),
activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3),
activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Flatten())
model.add(layers.Dense(64,
activation='relu'))
model.add(layers.Dense(62,
activation='softmax'))
print("Layers added")
model.summary()
```

Input shape: (100, 100, 1)
Layers added
Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 98, 98, 32)	320
max_pooling2d (MaxPooling2D)	(None, 49, 49, 32)	0
conv2d_1 (Conv2D)	(None, 47, 47, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 23, 23, 64)	0
conv2d_2 (Conv2D)	(None, 21, 21, 64)	36928
max_pooling2d_2 (MaxPooling2D)	(None, 10, 10, 64)	0
conv2d_3 (Conv2D)	(None, 8, 8, 64)	36928
max_pooling2d_3 (MaxPooling2D)	(None, 4, 4, 64)	0
flatten (Flatten)	(None, 1024)	0
dense (Dense)	(None, 64)	65600
dense_1 (Dense)	(None, 62)	4030
=====		
Total params: 162,302		
Trainable params: 162,302		
Non-trainable params: 0		