2019 Machine Learning II Kannada Digits Recognition

Junchi Tian

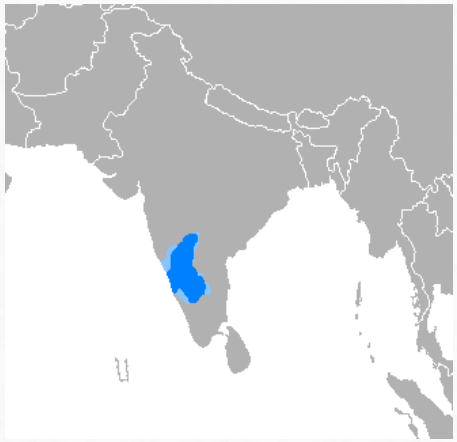
Jingshu Song

Yiming Dong

Outline

- Introduction
- Description of dataset
- Network and training algorithm
- Experimental setup
- Results
- Summary and conclusions

Introduction



- Kannada is a language spoken predominantly by people of Karnataka in southwestern India.
- The language has roughly 45 million native speakers and is written using the Kannada script.

Description of Dataset

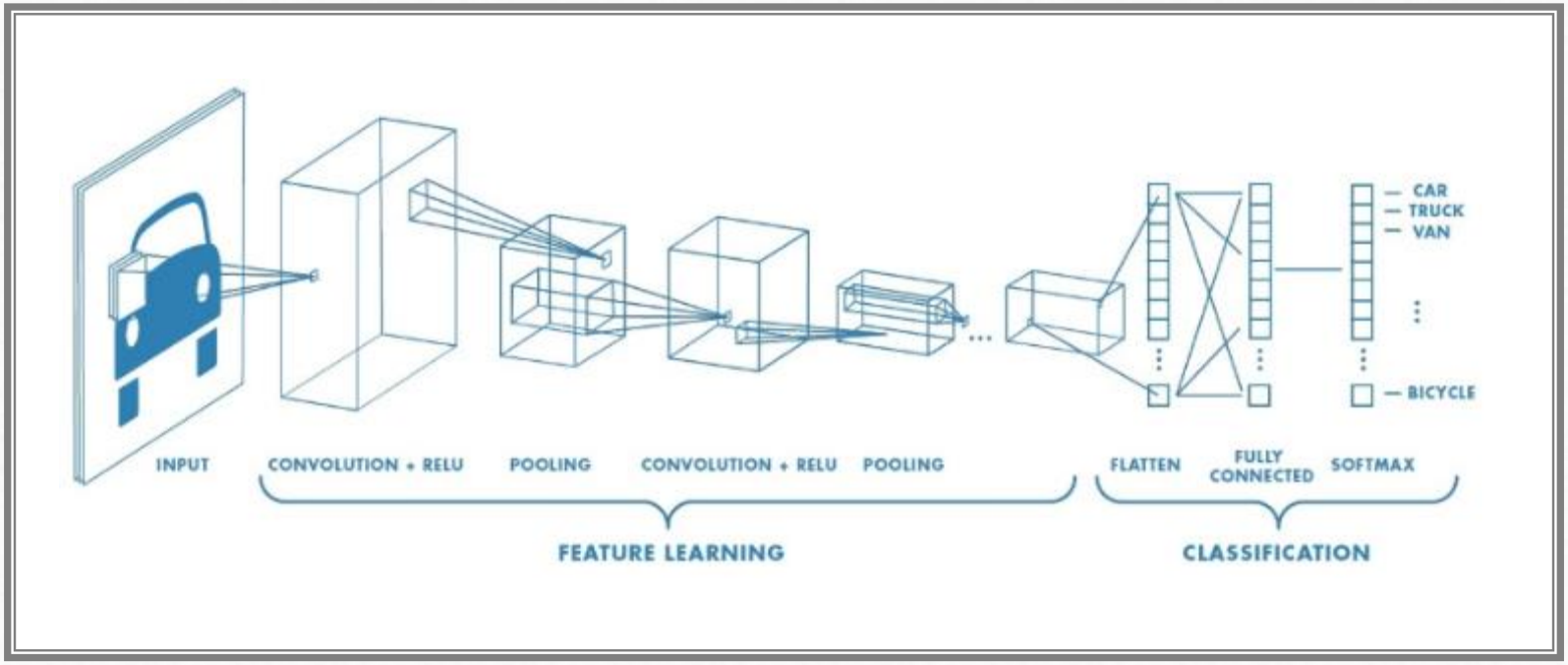
- This dataset is similar to the most famous computer vision dataset: handwritten digits recognition provided by Yann LeCun. Instead of using Arabic digits, this dataset uses Kannada digits. So, this is multi-classification model and will be used to recognize Kannada numeric from 0 to 9.

Kannada Kannada Digits from 1 to 10

Description of Dataset

- **Kannada-MNIST dataset**: 28X 28 grayscale images: 60k Train | 10k Test
- Raw Data can be accessed from Kaggle.com : https://www.kaggle.com/c/Kannada-MNIST

	A	B	C	D	E	F	G	H	I
1	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7
2	0	0	0	0	0	0	0	0	0
3	1	0	0	0	0	0	0	0	0
4	2	0	0	0	0	0	0	0	0
5	3	0	0	0	0	0	0	0	0
6	4	0	0	0	0	0	0	0	0
7	5	0	0	0	0	0	0	0	0
8	6	0	0	0	0	0	0	0	0
9	7	0	0	0	0	0	0	0	0
10	8	0	0	0	0	0	0	0	0
11	9	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0
13	1	0	0	0	0	0	0	0	0



The Framework and deep network

Keras + CNN

```
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = 0.20, r
andom_state=42)
x_train = x_train.values.reshape(-1,28,28,1)/255.0
x_test = x_test.values.reshape(-1,28,28,1)/255.0
y_train = np_utils.to_categorical(y_train,num_classes=10)
y_test = np_utils.to_categorical(y_test,num_classes=10)
```

Data Preprocessing

Image Generator

```
imagegen = ImageDataGenerator(
            rotation_range = 20,
            width_shift_range = 0.2,
            height_shift_range = 0.2,
            shear_range = 0.1,
            zoom_range = 0.2
             )
```

Modeling

```
model = Sequential()

model.add(Conv2D(input_shape=(28, 28, 1),filters=64, kernel_size=(3, 3), padd
ing='SAME', activation='relu'))
model.add(Conv2D(64, kernel_size=(3, 3), padding='SAME', activation='relu'))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

model.add(Conv2D(filters=128, kernel_size=(3, 3), padding='SAME', activation=
'relu'))
model.add(Conv2D(filters=128, kernel_size=(3, 3), padding='SAME', activation=
'relu'))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

model.add(Conv2D(filters=256, kernel_size=(3, 3), padding='SAME', activation=
'relu'))
model.add(Conv2D(filters=256, kernel_size=(3, 3), padding='SAME', activation=
'relu'))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

model.add(Flatten())
model.add(Dense(256, activation = "relu"))
model.add(BatchNormalization())
model.add(Dense(10, activation = "softmax"))
```

Fix CNN Overfitting

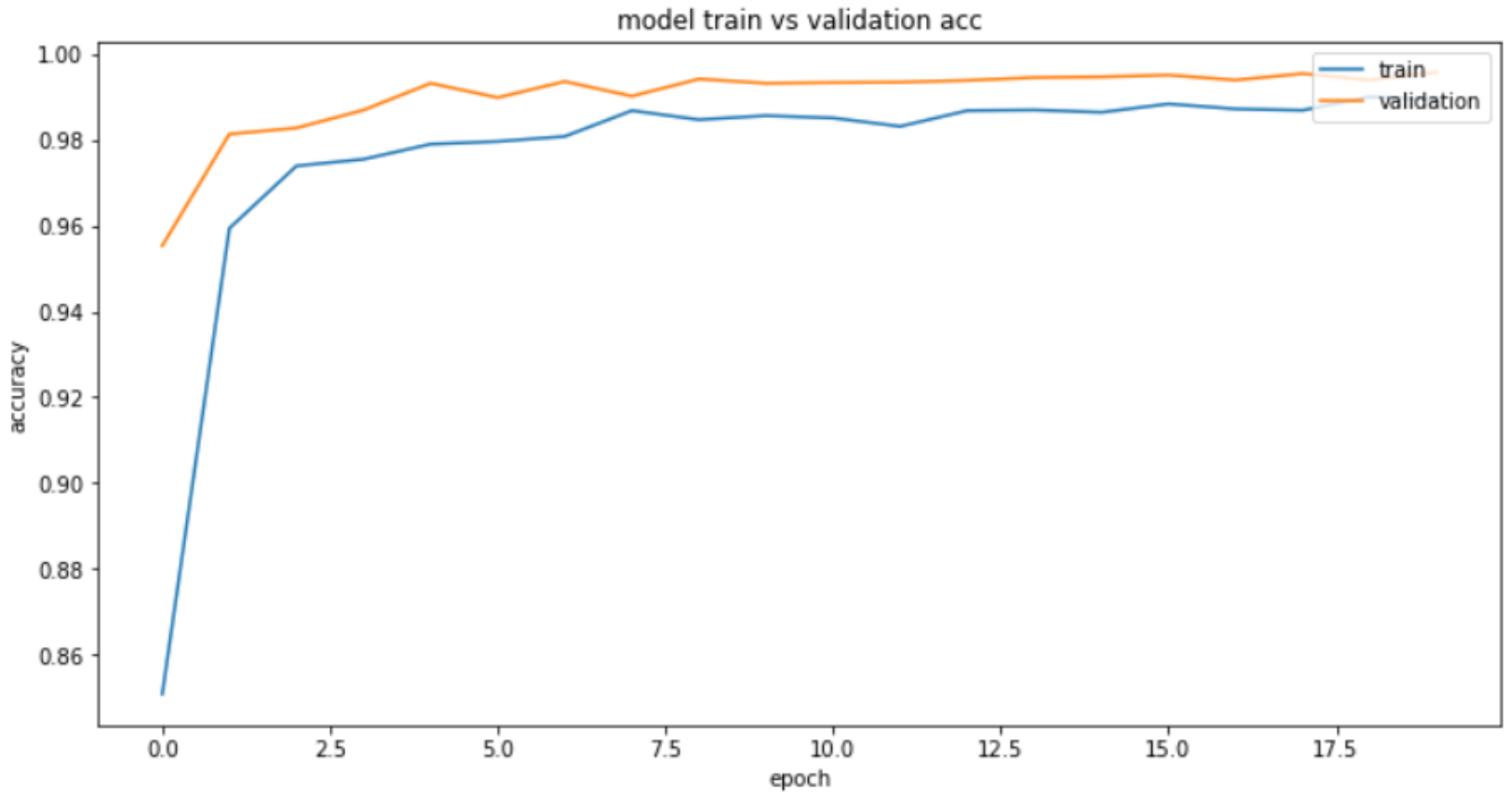
```
model.add(Conv2D(input_shape=(28, 28, 1),filters=64, kernel_size=(3, 3), padd
ing='SAME', activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(64, kernel_size=(3, 3), padding='SAME', activation='relu'))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.2))
```

Remove BatchNomalization

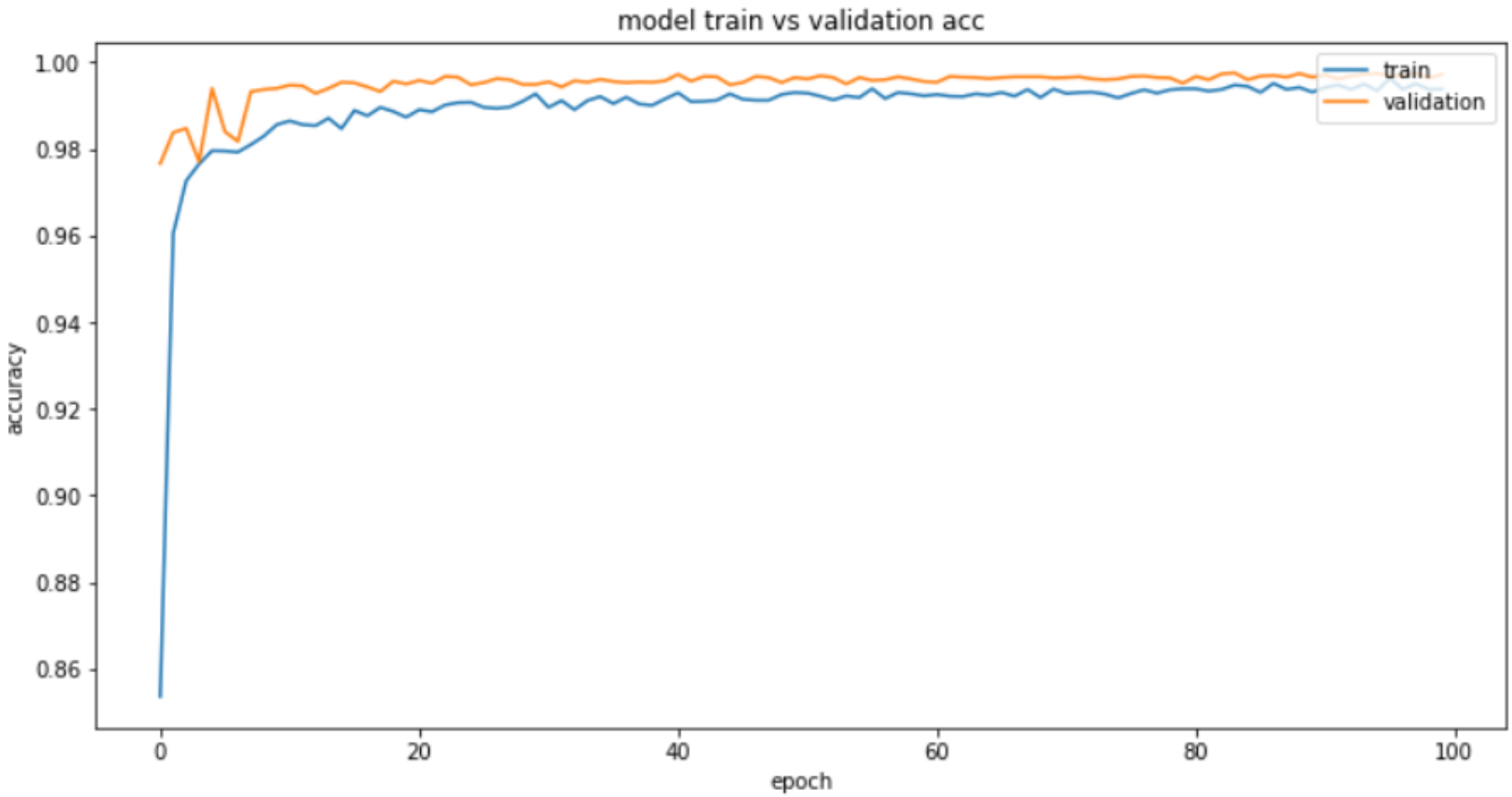
Learning Rate Reduction

```
learning_rate_reduction = ReduceLROnPlateau(monitor='val_acc',
                                            patience=3,
                                            verbose=1,
                                            factor=0.5,
                                            min_lr=0.00001)
```

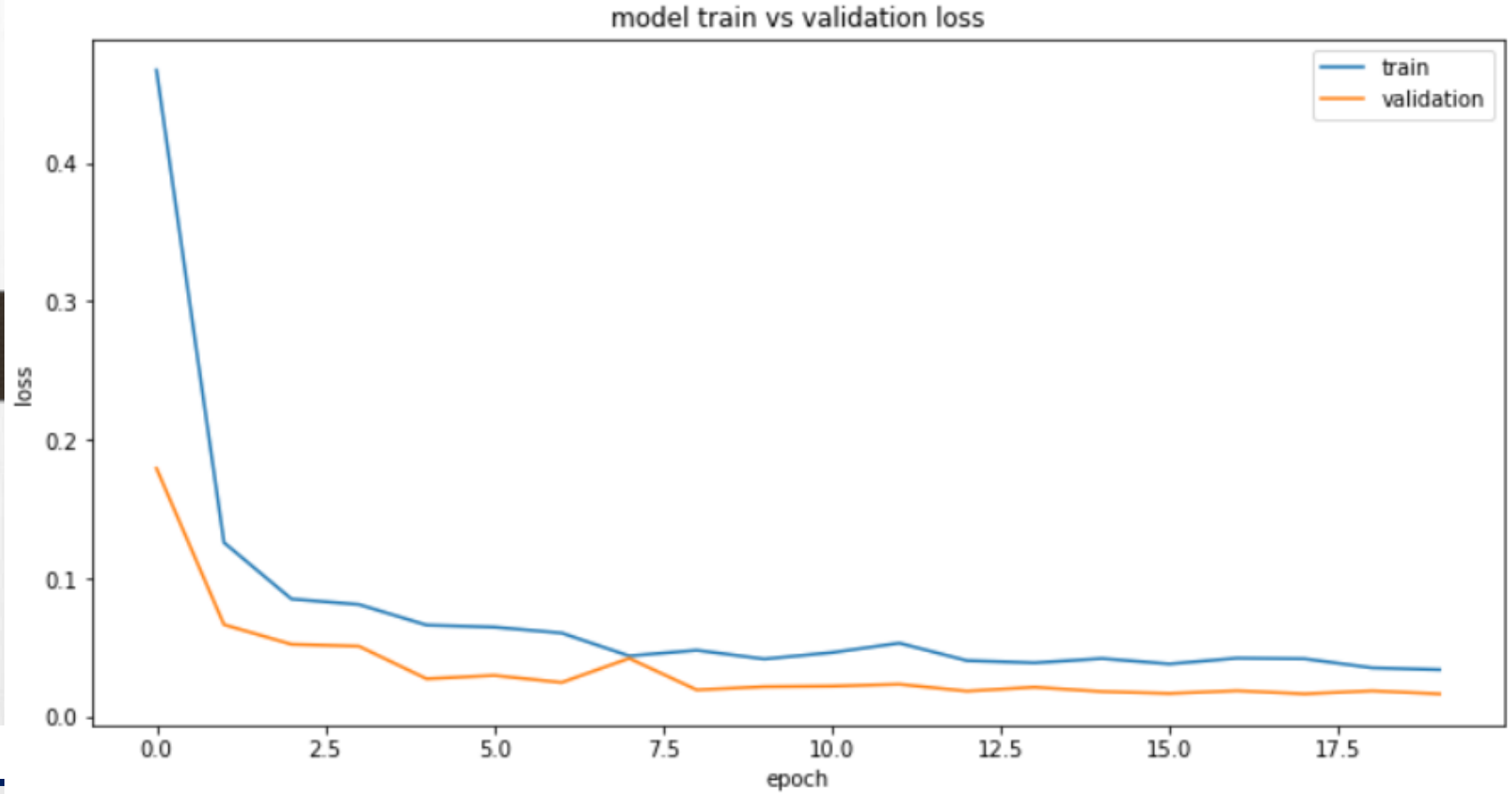
Results - Accuracy



Results – Accuracy (Improved)



Results - Loss



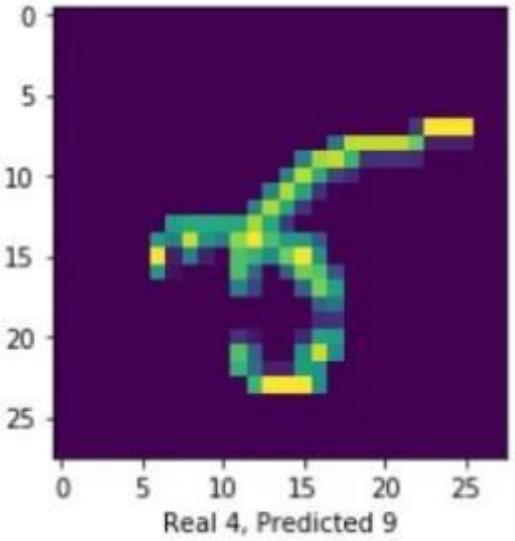
Results – Loss (Improved)

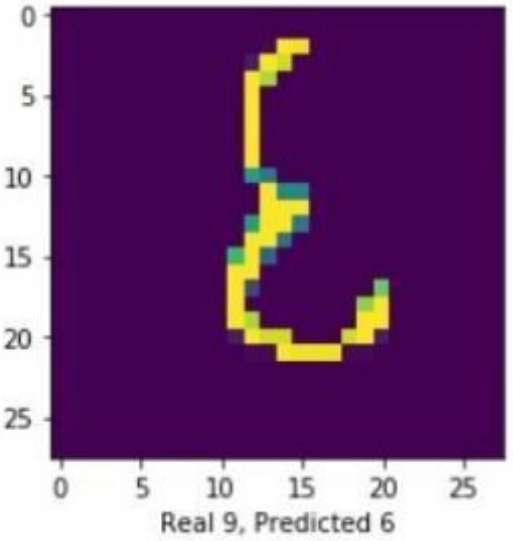
Final Results

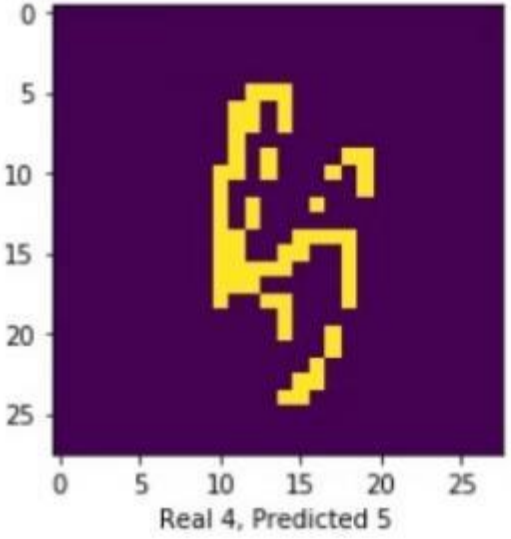
	0	1	2	3	4	5	6	7	8	9
0	1167	10	0	0	0	0	0	0	0	0
1	3	1215	0	0	0	0	0	0	0	0
2	1	0	1223	0	0	0	0	0	0	0
3	0	0	0	1174	0	0	0	10	0	0
4	0	1	0	0	1220	0	0	0	0	0
5	0	0	0	1	0	1187	0	0	0	0
6	0	0	0	0	0	0	1167	2	0	0
7	0	0	1	0	0	0	2	1216	0	0
8	0	0	0	0	0	0	0	0	1186	0
9	0	0	0	0	1	0	13	0	0	1200

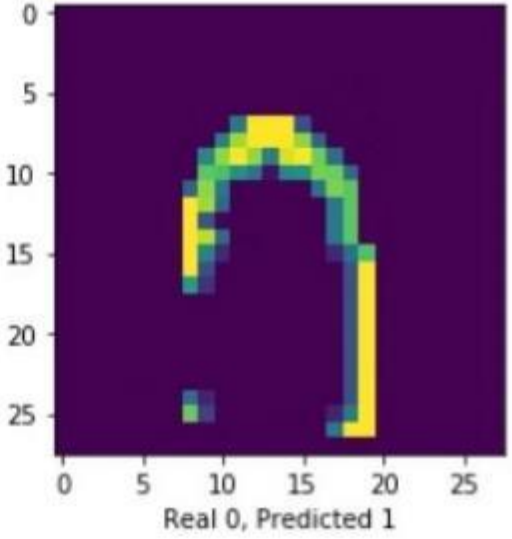
```
test loss 0.0141626347177055 62
test accuracy 0.9962499737739563
```

Prediction


0
5
10
15
20
25
Real 4, Predicted 9


0
5
10
15
20
25
Real 9, Predicted 6


0
5
10
15
20
25
Real 4, Predicted 5


0
5
10
15
20
25
Real 0, Predicted 1

Thank You