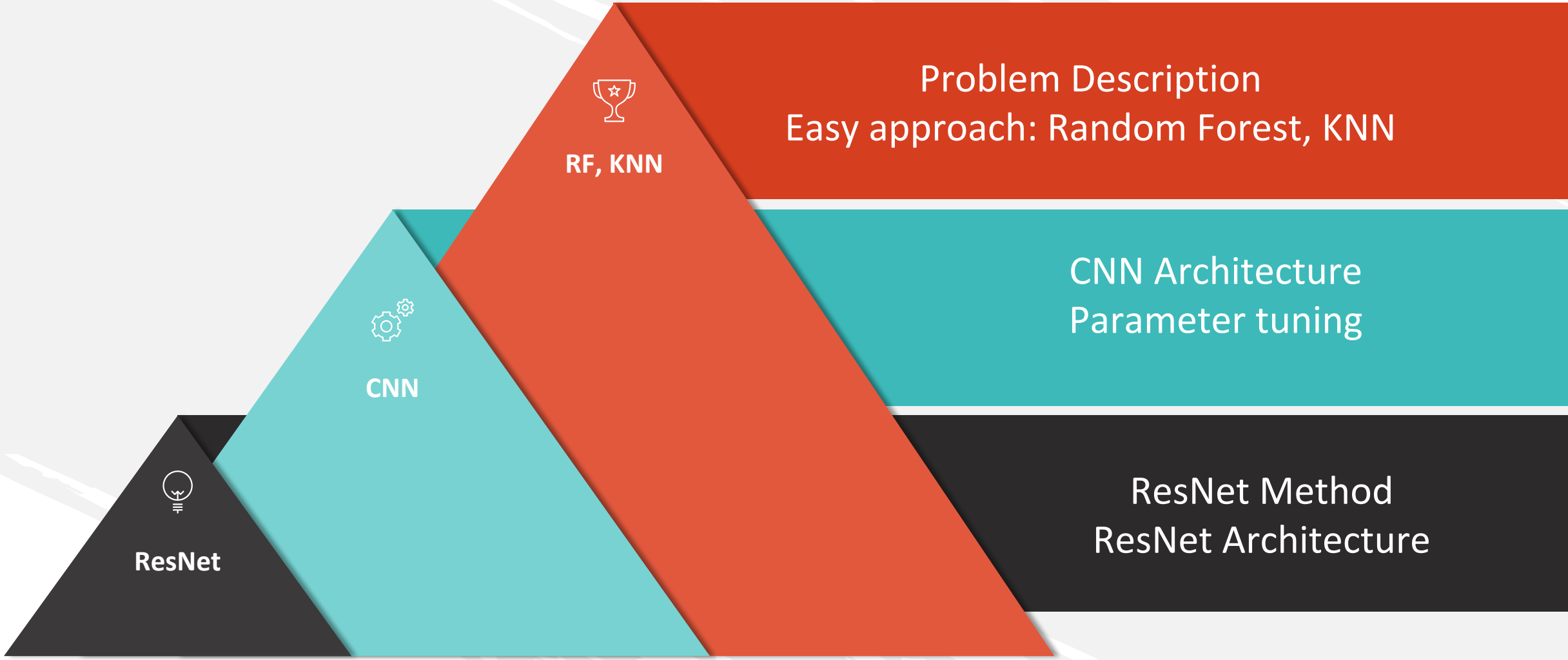
CNN

Final Project Presentation

Brett Castello | Yiming Gu | Shenghua Yue

Contents


RF, KNN
CNN
ResNet
Problem Description
Easy approach: Random Forest, KNN
CNN Architecture
Parameter tuning
ResNet Method
ResNet Architecture

Problem Description

Kaggle Digit Recognizer Competition: Learn computer vision fundamentals with the famous MNIST data. (https://www.kaggle.com/c/digit-recognizer)

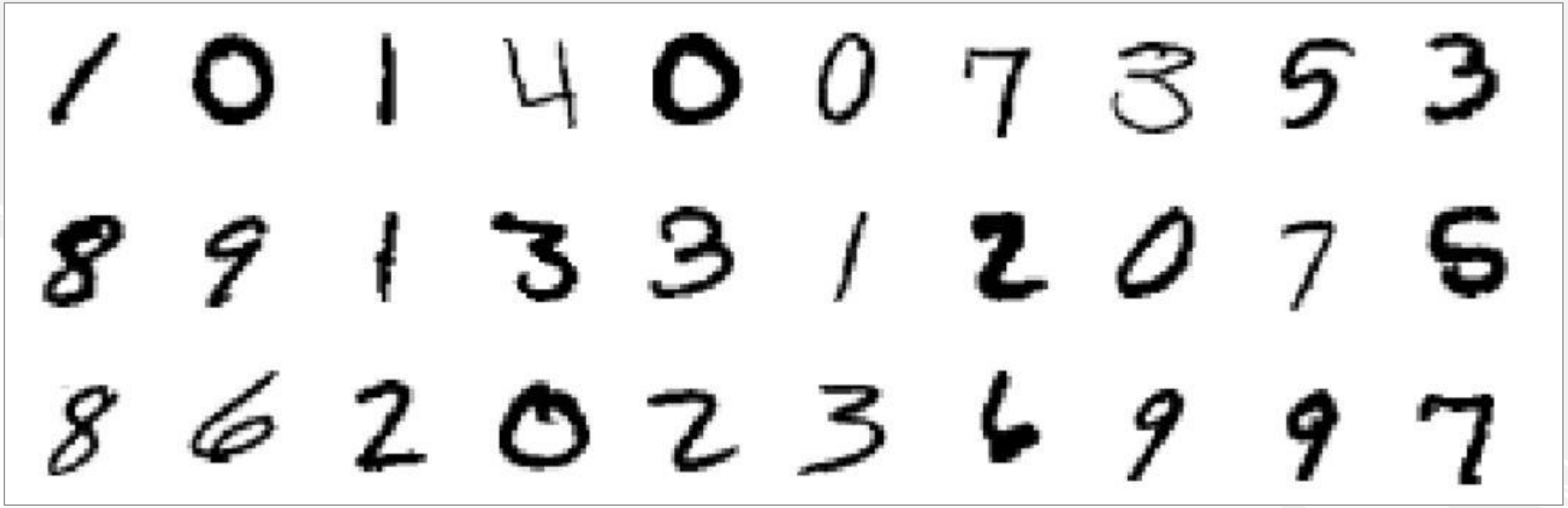
MNIST data released in 1999, this classic dataset of handwritten images has served as the basis for benchmarking classification algorithms. It is a reliable resource for researchers and learners.

Our goal is to correctly identify digits from a dataset of tens of thousands of handwritten images. The training data includes 42000 digits. The training data are used to train the model and to predict 28000 digits in the test data.

Problem Description

The dataset consists of pair, “handwritten digit image” and “label”. Digit ranges from 0 to 9, meaning 10 patterns in total.

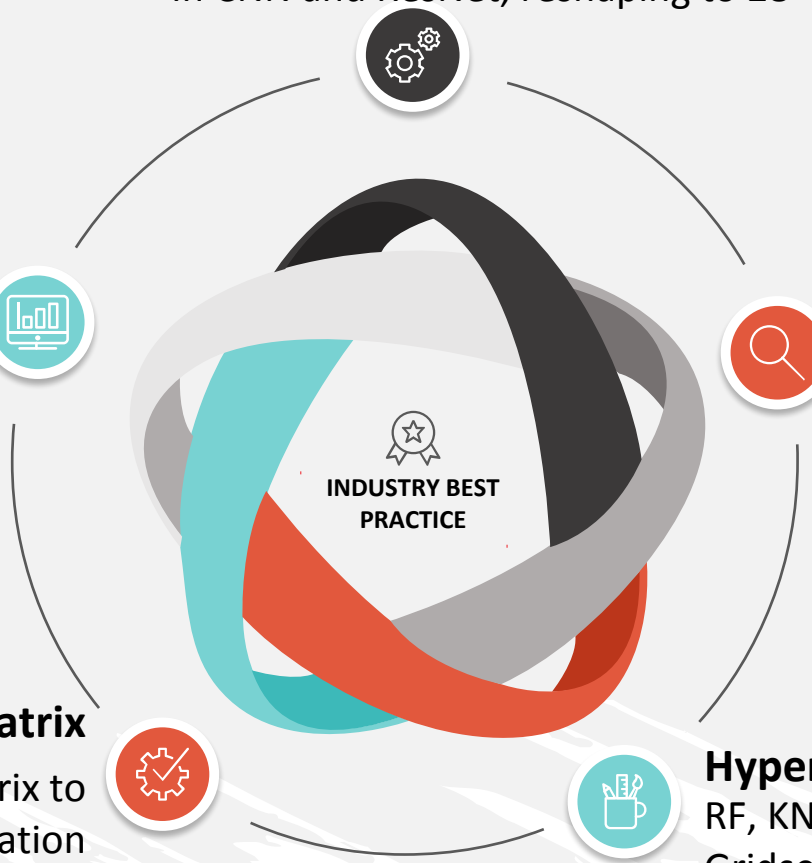
- handwritten digit image: This is gray-scale image with size 28 x 28 pixel (784 columns).
- label : This is actual digit number this handwritten digit image represents (1 column). It is either 0 to 9.



Overview

Feature Engineering

Performing a grayscale normalization to reduce effect of illumination differences and model converges faster
In CNN and ResNet, reshaping to 28*28



Architecture

CNN, ResNet
convolution layer, pooling, batch normalization, dense layer, dropout

Hyperparameters

RF, KNN, CNN, ResNet
Gridsearch

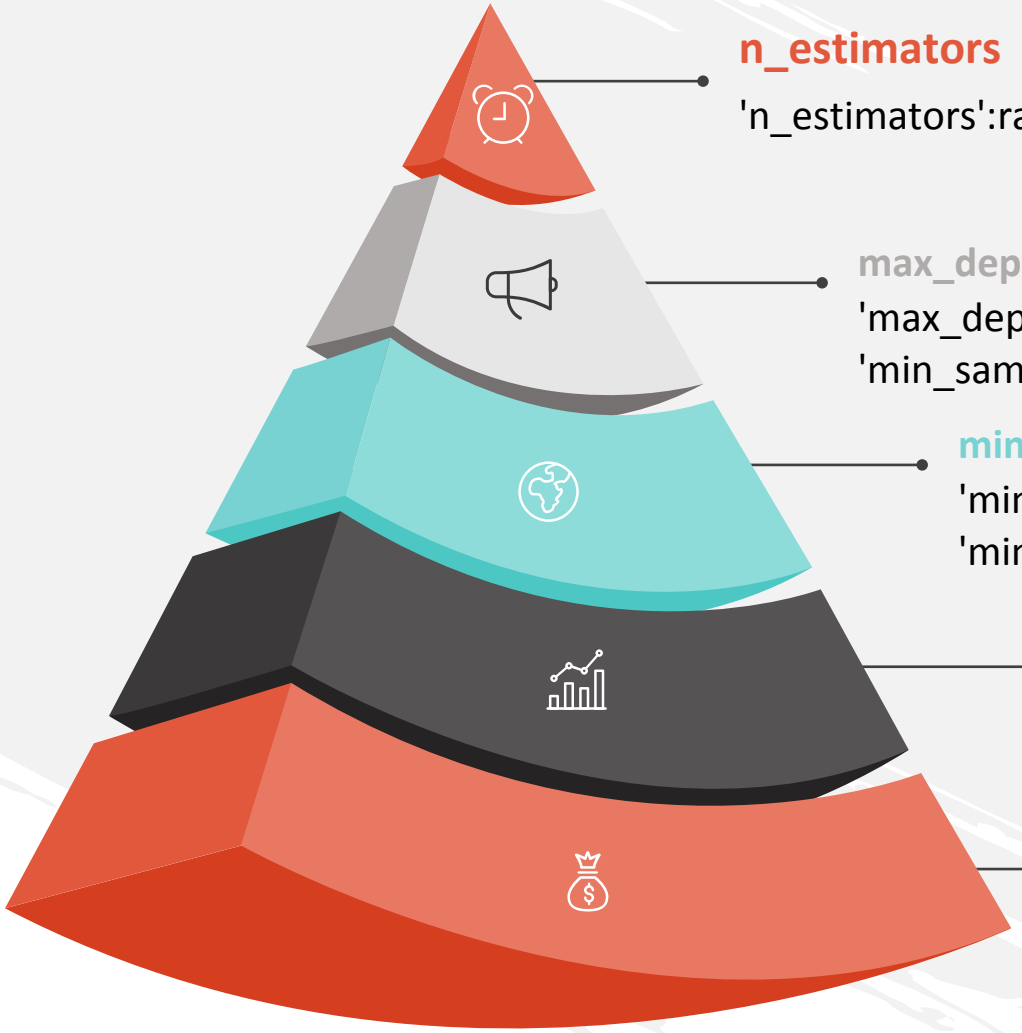
Confusion Matrix

Using confusion matrix to evaluate multi-classification results

Results Testing

Prediction results of 28000 digits are tested in Kaggle

Hyperparameter Tuning in Random Forest



n_estimators

'n_estimators':range(50,650,100)

max_depth & min_samples_split

'max_depth':range(5,21,5)
'min_samples_split':range(50,201,50)

min_samples_leaf & min_samples_split

'min_samples_split':range(30,71,10)
'min_samples_leaf':range(10,51,10)

Lmin_samples_leaf & min_samples_split

'min_samples_split':range(2,31,5)
'min_samples_leaf':range(1,21,5)

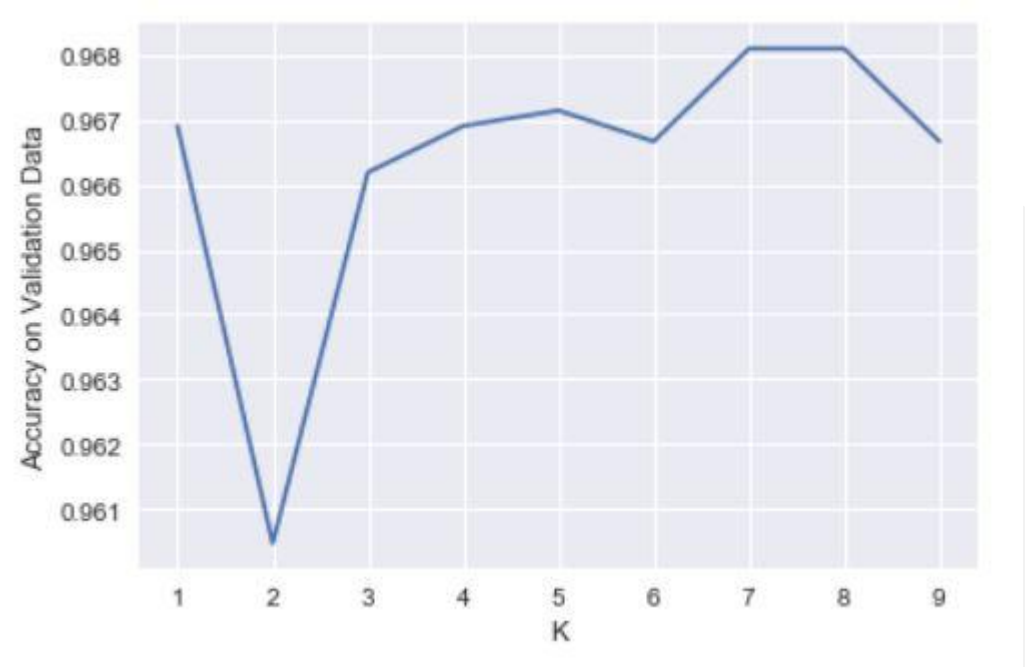
max_features

'max_features':['auto','sqrt','log2']

Random Forest & KNN

```
# train the model using best parameters
clf_RF = RandomForestClassifier(n_estimators= 500, oob_score = True,
                                max_depth=20, min_samples_split=2,min_samples_leaf=1,
                                max_features='auto', random_state=0)
clf_RF.fit(X_train, Y_train)
print("accuracy on training set: %f" % clf_RF.score(X_train, Y_train))
print("accuracy on test set: %f" % clf_RF.score(X_val, Y_val))
print("out-of-bag estimate: %f" % clf_RF.oob_score_)
```

```
accuracy on training set: 0.999656
accuracy on test set: 0.965476
out-of-bag estimate: 0.964815
```




```
knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(X_train, Y_train)
y_pred_cnn = knn.predict(X_val)
accuracy = (y_pred_knn == Y_val).mean()
accuracy
```

```
0.9666666666666667
```

Random Forest & KNN

```
# visualizing confusion matrix
Y_pred = clf_RF.predict(X_val)
cm = confusion_matrix(Y_val, Y_pred)
print(cm)
```

```
[[406   0   0   0   1   0   2   0   3   0]
 [  0 439   1   1   1   0   0   1   0   1]
 [  3   0 402   0   3   0   1   6   3   1]
 [  1   0   6 391   0   5   0   2   2   2]
 [  0   1   0   0 408   0   4   0   1  13]
 [  0   1   0   4   0 401   1   0   3   3]
 [  4   1   0   0   0   3 437   0   0   0]
 [  0   1   5   1   1   0   0 406   2   6]
 [  0   3   2   5   1   2   2   0 390   8]
 [  1   3   0   6   4   0   0   5   2 375]]
```

```
# visualizing confusion matrix
cm = confusion_matrix(Y_val, y_pred_knn)
print(cm)
```

```
[[430   0   0   0   0   0   2   0   0   1]
 [  0 467   1   0   0   0   0   2   0   0]
 [  5   4 402   1   1   1   0   6   1   0]
 [  0   2   2 402   0   3   0   3   2   1]
 [  0   6   0   0 376   0   1   1   0   7]
 [  2   0   0   4   2 366   3   0   1   1]
 [  3   1   0   0   0   1 397   0   0   0]
 [  0   9   1   0   1   0   0 438   0   5]
 [  1   6   0   6   1   6   2   0 385   6]
 [  1   2   1   1   5   0   1  13   1 397]]
```

Name	Submitted	Wait time	Execution time	Score
RF_results.csv	a few seconds ago	0 seconds	0 seconds	0.96528

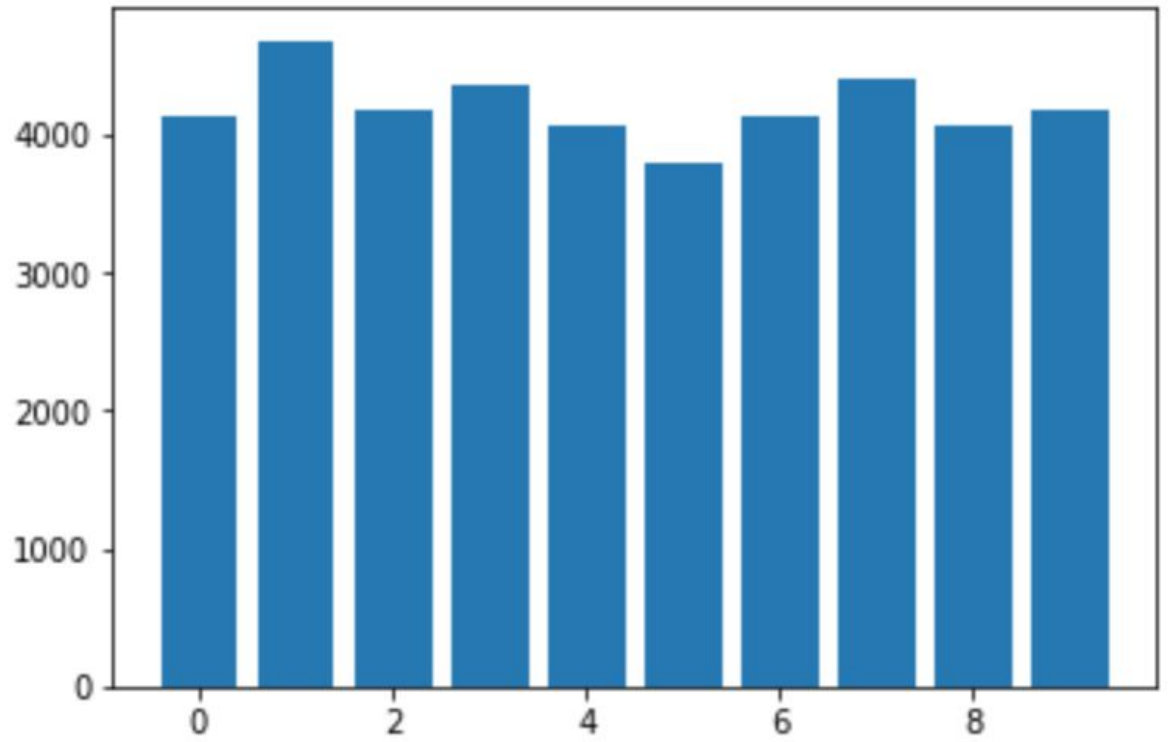
Name	Submitted	Wait time	Execution time	Score
KNN_results.csv	just now	0 seconds	0 seconds	0.96600

CNN Model Data Preprocessing

Check Label Distribution

```
# check if labels are equally distributed, if not, weights should be assigned
plt.bar(y_train.value_counts().index, y_train.value_counts().values)
```

```
<BarContainer object of 10 artists>
```



CNN Model Data Preprocessing

Normalization and Reshape

```
# preparing data
y_train = train["label"]
x_train = train.iloc[:,1:]
x_train = x_train / 255.0  # data range [0, 1]
x_test = test / 255.0      # data range [0, 1]
x_train = x_train.values.reshape(-1,28,28,1)   # input image dimensions: 28*28*1
x_test = x_test.values.reshape(-1,28,28,1)
y_train = keras.utils.to_categorical(y_train, num_classes = 10)  # convert a class vector (integers) to binary class ma
print('training data shape', x_train.shape)
print('testing data shape', x_test.shape)
print('training result shape', y_train.shape)
print('training result example', y_train[0])
```

```
training data shape (42000, 28, 28, 1)
testing data shape (28000, 28, 28, 1)
training result shape (42000, 10)
training result example [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
```

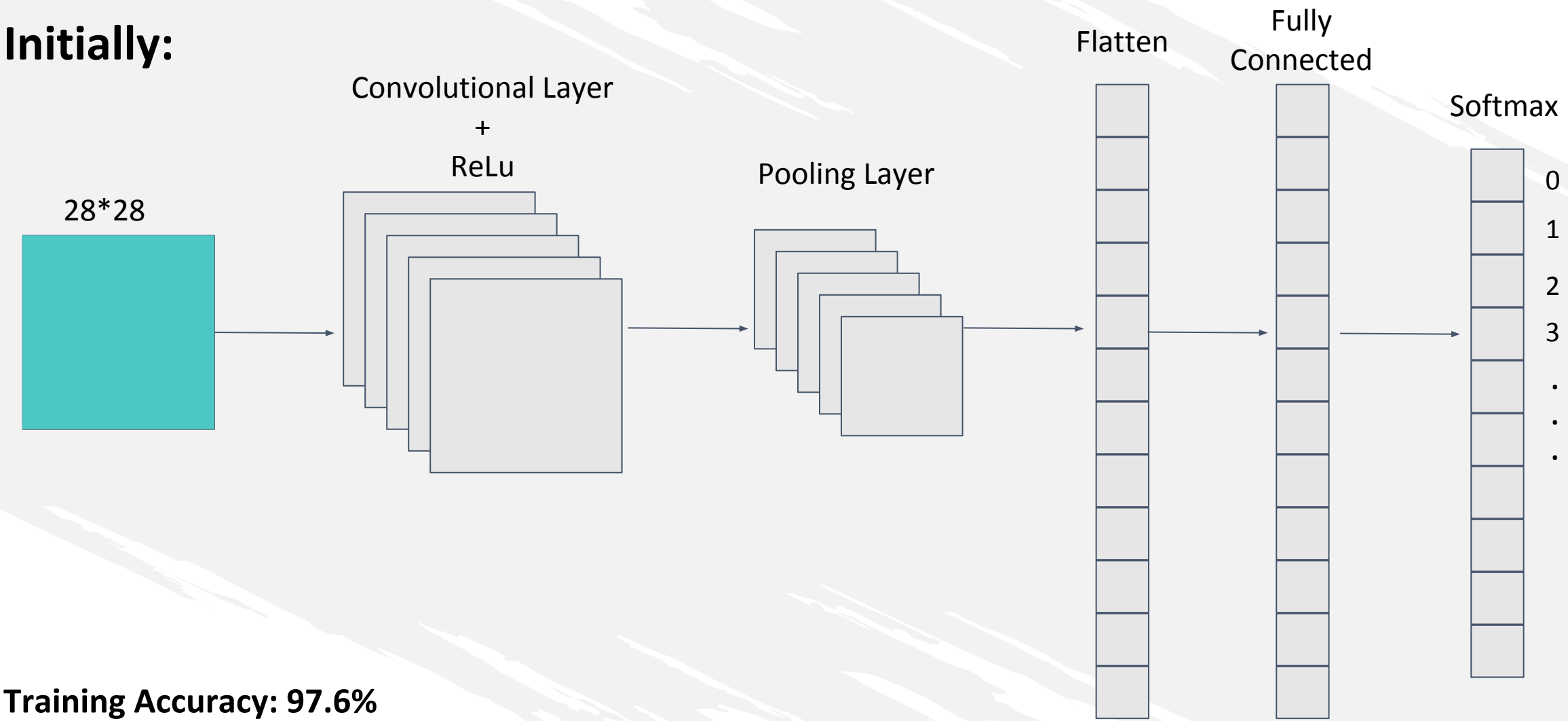
CNN Model Data Preprocessing

Split the training data set into TRAIN (90%, 37800) and VAL (10%, 4200)

```
# preparing training and testing data for model validation
X_train, X_val, Y_train, Y_val = train_test_split(x_train, y_train, test_size = 0.1)
```

CNN Model Architecture Design

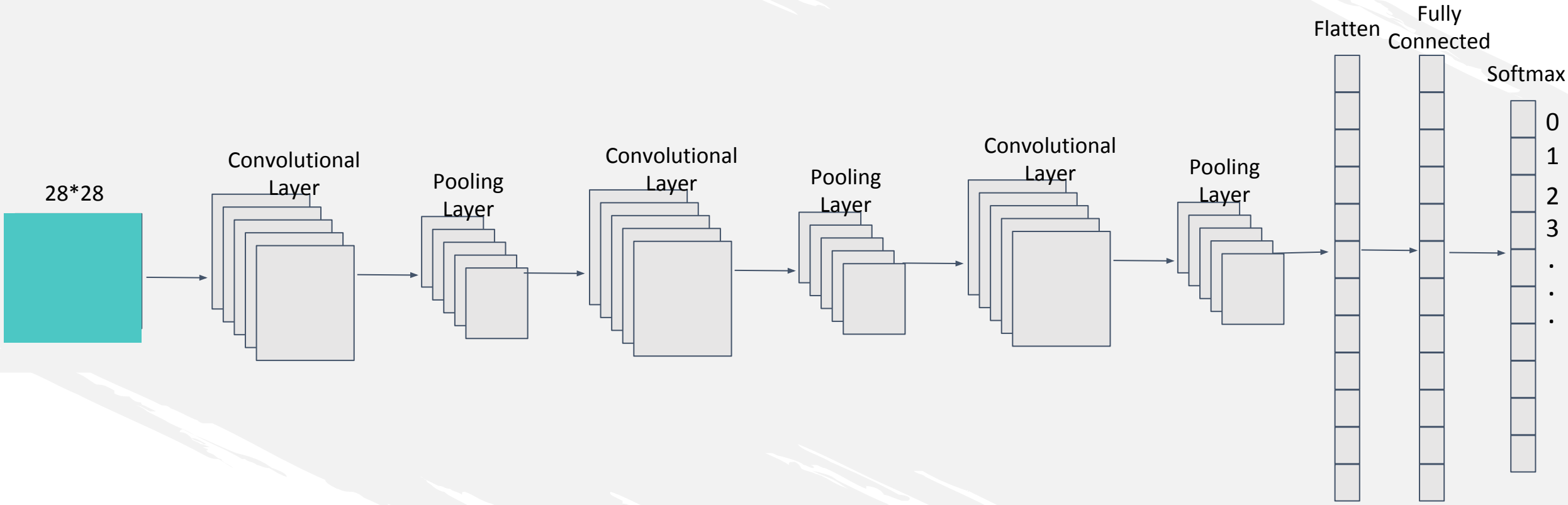
Initially:



Training Accuracy: 97.6%

CNN Model Architecture Design

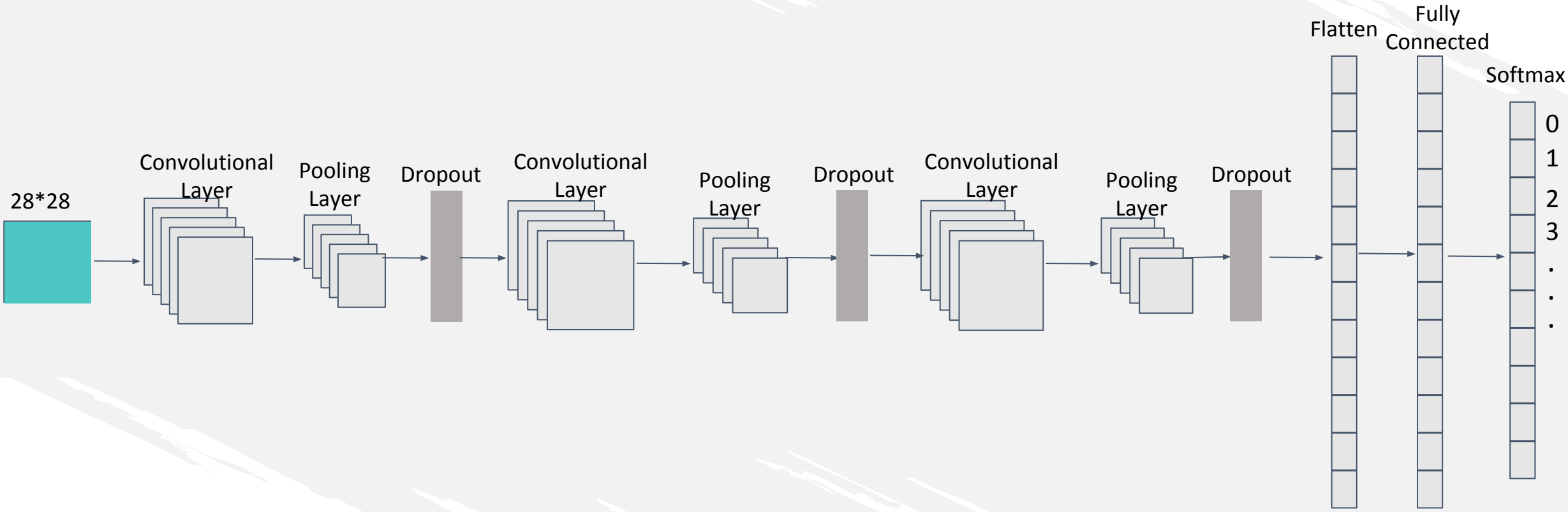
More Hidden Layers:



Training Accuracy: 99.2%

CNN Model Architecture Design

Add Dropout Layers:



Training Accuracy: 99.5%

CNN Model Architecture Design

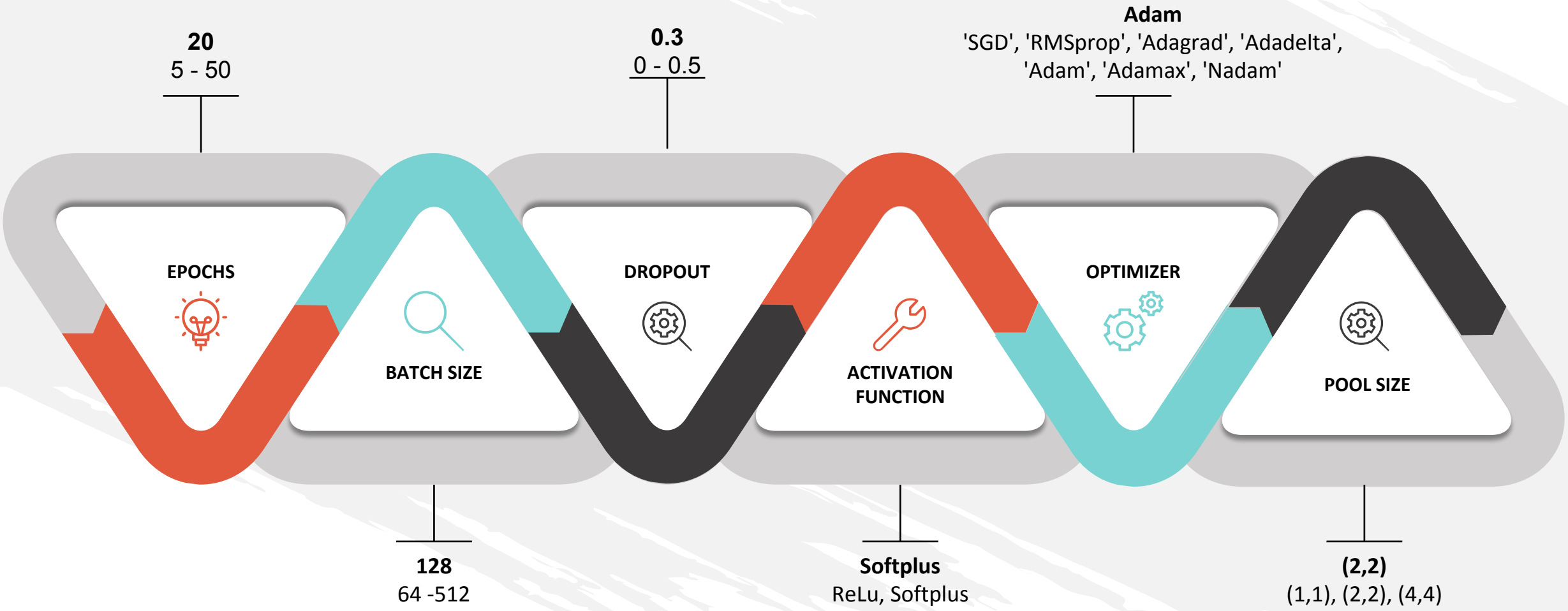
```
model = Sequential()
model.add(Conv2D(filters=16, kernel_size=3, activation='relu', input_shape=(28,28,1)))
model.add(BatchNormalization())
model.add(Conv2D(16, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.3))

model.add(Conv2D(filters=32, kernel_size=3, activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(32, 3, activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.3))

model.add(Conv2D(filters=64, kernel_size=3, activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(64, 3, padding = 'same', activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.3))

model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
```

CNN Model Hyperparameter Tuning and Cross Validation



CNN Model Hyperparameter Tuning and Cross Validation

Use the Wrapper for Scikit-Learn API in Keras to wrap the model:

`keras.wrappers.scikit_learn.KerasClassifier(build_fn=None, **sk_params)`

```
# choosing optimization algorithms
batch_size = 128
epochs = 20
num_classes = 10
optimizer = 'Adam'
pool_size = 2
dropout = 0.3
activation = 'relu'

clf = KerasClassifier(build_fn=cnn_model, batch_size=batch_size, epochs=epochs, \
                      verbose=1,optimizer=optimizer,pool_size=pool_size,dropout=dropout,activation=activation)

optimizer = ['SGD', 'RMSprop', 'Adagrad', 'Adadelta', 'Adam', 'Adamax', 'Nadam']

param_grid = dict(optimizer=optimizer)
grid = GridSearchCV(estimator=clf, param_grid=param_grid, n_jobs=-1, verbose=1)
grid_result = grid.fit(X_train, Y_train)
```

CNN Model Hyperparameter Tuning and Cross Validation

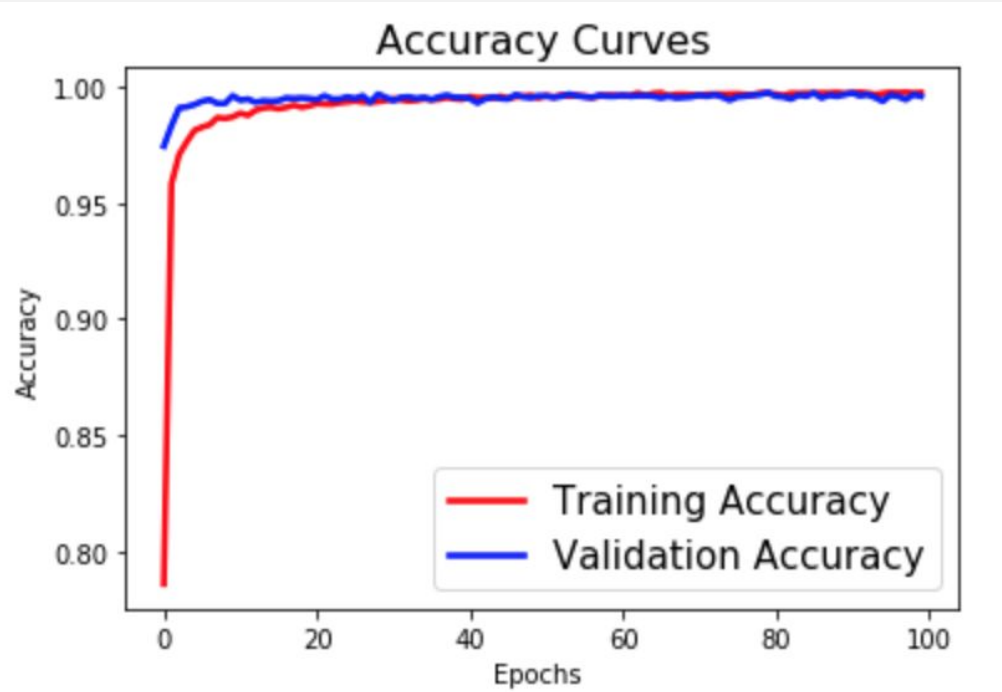
```
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

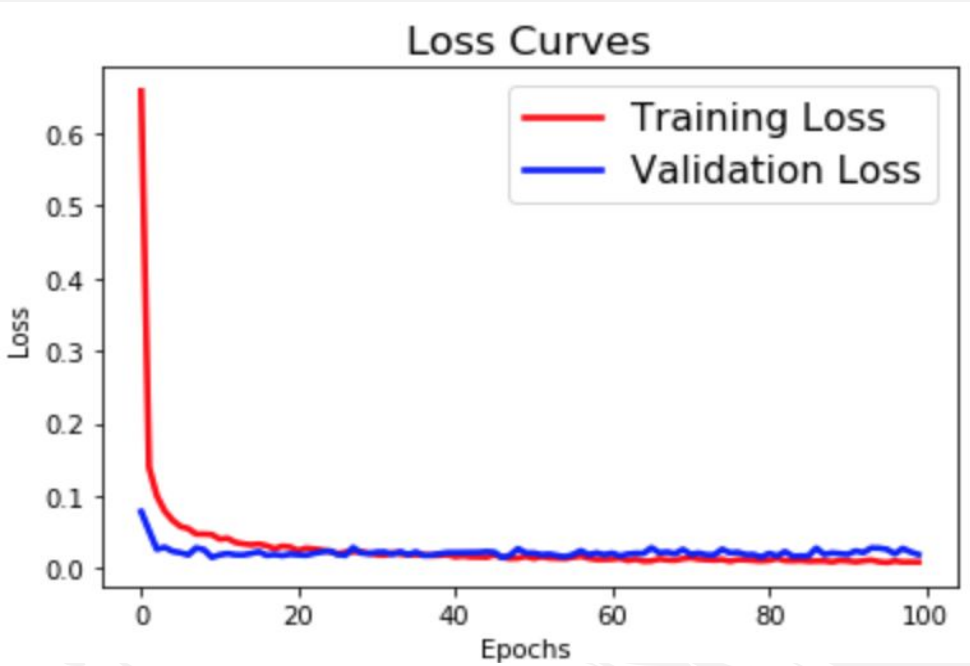
```
Best: 0.991296 using {'optimizer': 'Adam'}
0.985767 (0.000714) with: {'optimizer': 'SGD'}
0.990582 (0.001673) with: {'optimizer': 'RMSprop'}
0.990476 (0.000405) with: {'optimizer': 'Adagrad'}
0.991085 (0.000491) with: {'optimizer': 'Adadelta'}
0.991296 (0.000944) with: {'optimizer': 'Adam'}
0.991138 (0.001722) with: {'optimizer': 'Adamax'}
0.990185 (0.000551) with: {'optimizer': 'Nadam'}
```

CNN Model Fitting

```
# fit the final model
finalCNN = cnn_model(optimizer = 'Adam', pool_size = 2, dropout = 0.3, activation = 'softplus')

CNN = finalCNN.fit(X_train, Y_train,
          batch_size=128,
          epochs=100,
          verbose=1,
          validation_data=(X_val, Y_val))
```





CNN Model Evaluation

```
# confusion_matrix
Y_model = finalCNN.predict(X_val)
Y_pred = np.argmax(Y_model, axis=1)
Y_true = np.argmax(Y_val, axis=1)
confusion_matrix(Y_true, Y_pred)
```

```
array([[428,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0, 436,   0,   0,   0,   0,   0,   0,   1,   0],
       [  0,   0, 419,   0,   0,   0,   0,   0,   1,   0],
       [  0,   0,   0, 445,   0,   0,   0,   0,   1,   0],
       [  0,   0,   0,   0, 378,   0,   1,   0,   0,   1],
       [  0,   0,   0,   0,   0, 413,   0,   0,   1,   1],
       [  0,   0,   1,   0,   1,   0, 435,   0,   0,   0],
       [  0,   0,   0,   0,   0,   0,   0, 447,   1,   0],
       [  0,   2,   1,   0,   0,   0,   1,   0, 387,   0],
       [  2,   0,   0,   0,   2,   0,   0,   0,   0, 394]])
```

```
# Evaluation
score = finalCNN.evaluate(X_val, Y_val, verbose=0)
print('Val loss:', score[0])
print('Val accuracy:', score[1])
```

```
Val loss: 0.01915301335235 34
Val accuracy: 0.9957142857142857
```

```
# other metrics
print('Precision Score:',precision_score(Y_true, Y_pred, average='weighted'))
print('Recall Score:',recall_score(Y_true, Y_pred,average='weighted'))
print('F1 Score:',f1_score(Y_true, Y_pred,average='weighted'))
```

```
Precision Score: 0.9957212255326522
Recall Score: 0.9957142857142857
F1 Score: 0.9957151831419324
```

Name	Submitted	Wait time	Execution time	Score
CNN_100.csv	just now	0 seconds	0 seconds	0.99457

Complete

Data Augmentation

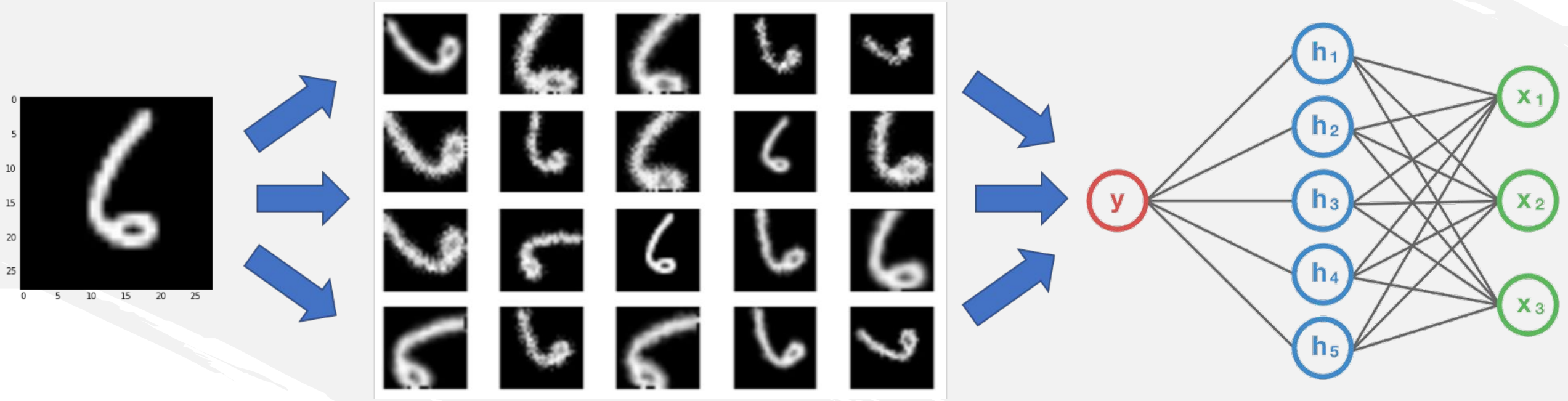


Image Reference: https://hazyresearch.github.io/snorkel/blog/tanda.html

CNN Model After Data Augmentation

```
imggen = ImageDataGenerator(featurewise_center = False,
                            samplewise_center = False,
                            featurewise_std_normalization = False,
                            samplewise_std_normalization = False,
                            zca_whitening = False,
                            rotation_range = 10,
                            zoom_range = 0.10,
                            width_shift_range = 0.10,
                            height_shift_range = 0.10,
                            horizontal_flip = False,
                            vertical_flip = False)

imggen.fit(X_train)
```

result_cnn_100epoch_regularized.csv 0.99728

9 hours ago by Condor76

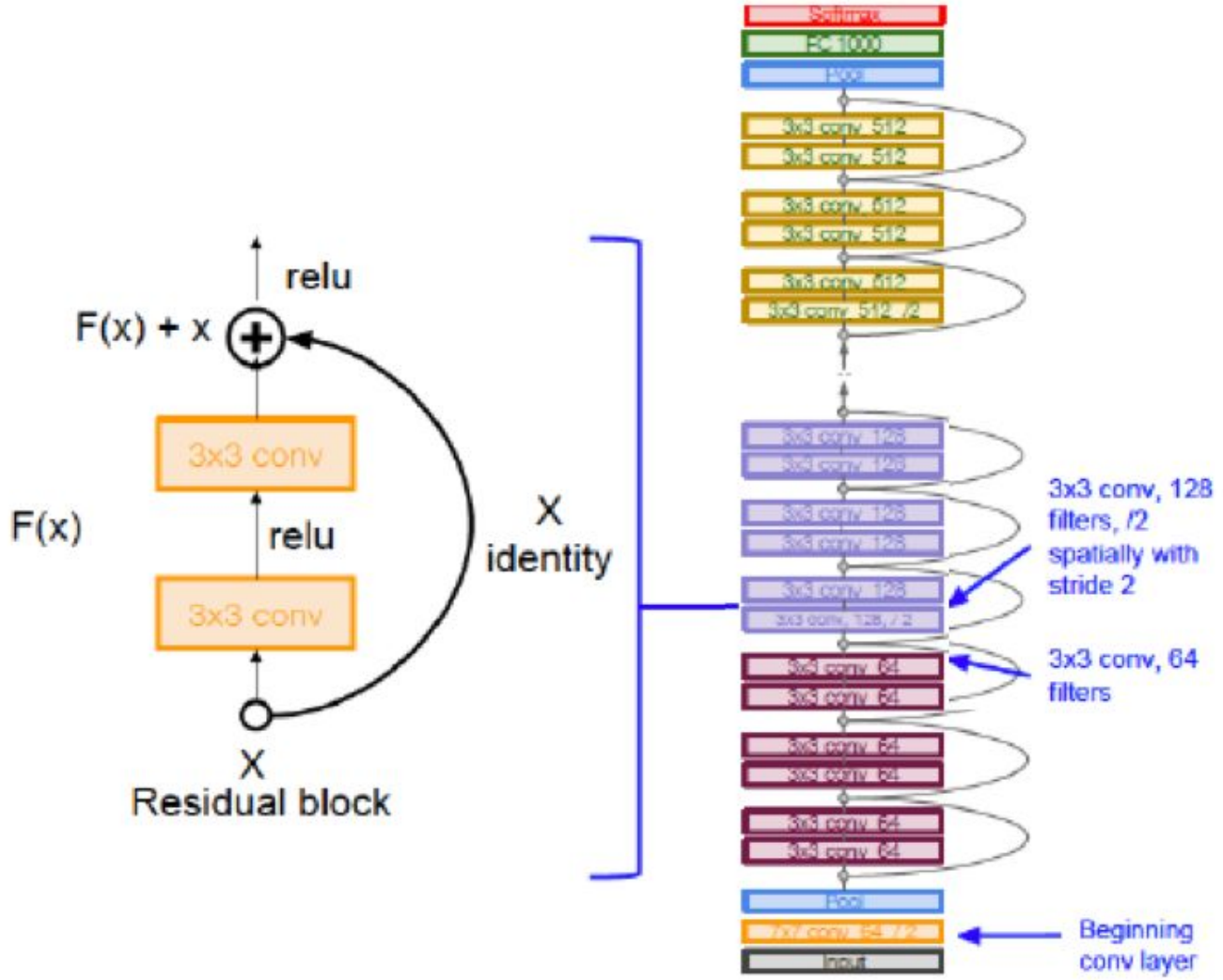
100 Epoch convolutional neural network with l2 regularization.

RESNET Model

- Solves vanishing gradients problem
- Adds reinforcement of signal after a few layers
- Modeled after cordical layer VI neurons in the brain

Source: https://www.kaggle.com/meownoid/tiny-resnet-with-keras-99-314

RESNET Architecture



RESNET Code

```
def block(n_output, upscale=False):
    # n_output: number of feature maps in the block
    # upscale: should we use the 1x1 conv2d mapping for shortcut or not

    # keras functional api: return the function of type
    # Tensor -> Tensor
    def f(x):

        # H_l(x):
        # first pre-activation
        h = BatchNormalization()(x)
        h = Activation(relu)(h)
        # first convolution
        h = Conv2D(kernel_size=3, filters=n_output, strides=1, padding='same

        # second pre-activation
        h = BatchNormalization()(x)
        h = Activation(relu)(h)
        # second convolution
        h = Conv2D(kernel_size=3, filters=n_output, strides=1, padding='same

        # f(x):
        if upscale:
            # 1x1 conv2d
            f = Conv2D(kernel_size=1, filters=n_output, strides=1, padding='
        else:
            # identity
            f = x

        # F_l(x) = f(x) + H_l(x):
        return add([f, h])

    return f
```

```
input_tensor = Input((28, 28, 1))

# first conv2d with post-activation to
x = Conv2D(kernel_size=3, filters=16,
x = BatchNormalization()(x)
x = Activation(relu)(x)

# F_1
x = block(16)(x)
# F_2
x = block(16)(x)

# F_3
# H_3 is the function from the tensor
# and we can't add together tensors of
x = block(32, upscale=True)(x)       #
# F_4
x = block(32)(x)                     #
# F_5
x = block(32)(x)                     #

# F_6
x = block(48, upscale=True)(x)       #
# F_7
x = block(48)(x)                     #


# F_8
x = block(64, upscale=True)(x)       #
# F_9
x = block(64)(x)

# F_8
x = block(128, upscale=True)(x)
# F_9
x = block(128)(x)

# last activation of the entire networ
x = BatchNormalization()(x)
x = Activation(relu)(x)

# average pooling across the channels
# 28x28x48 -> 1x48
x = GlobalAveragePooling2D()(x)

# dropout for more robust learning
x = Dropout(0.2)(x)

# last softmax layer
```

Model Comparison

	KNN	Random Forest	CNN	CNN with Data Augmentation	ResNet
Training Accuracy	0.9752	0.9997	0.9973	0.9981	0.9938
Validation Accuracy	0.9698	0.9655	0.9957	0.9964	0.7356
Precision Score	0.9701	0.9700	0.9957	0.9985	0.8380
Recall Score	0.9698	0.9700	0.9957	0.9985	0.8380
F1 Score	0.9697	0.9700	0.9957	0.9985	0.8380
Test Accuracy (Kaggle)	0.9660	0.9653	0.9946	0.9973	0.8111
Time to run	fast, 5 mins	fast, 5 mins	slow, 1.5 hr+	slow, 2 hrs +	slow, 1 hr +

Thank You
&
Questions