Lecture 09
Network Tuning II

仿生设计与学习实验室
Bionic Design & Learning Lab

AncoraSIR.com

SUSTech
Southern University of Science and Technology

A Few Optimization Algorithms

Nesterov Momentum

Add a correction factor to the standard method of momentum

Algorithm 8.2 Stochastic gradient descent (SGD) with momentum

Require: Learning rate ϵ, momentum parameter α

Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.

while stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.

Compute gradient estimate: $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$

end while

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \nabla_{\boldsymbol{\theta}} \left(\frac{1}{m} \sum_{i=1}^{m} L(\boldsymbol{f}(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), \boldsymbol{y}^{(i)}) \right), \qquad \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}.$$

Algorithm 8.3 Stochastic gradient descent (SGD) with Nesterov momentum

Require: Learning rate ϵ, momentum parameter α.

Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.

while stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ with corresponding labels $\boldsymbol{y}^{(i)}$.

Apply interim update: $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$

Compute gradient (at interim point): $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\boldsymbol{x}^{(i)};\tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$

Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$

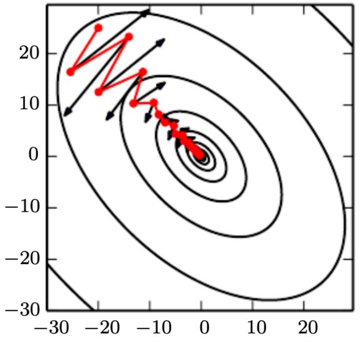
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$

end while

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \nabla_{\boldsymbol{\theta}} \left[\frac{1}{m} \sum_{i=1}^{m} L\left(\boldsymbol{f}(\boldsymbol{x}^{(i)}; \boxed{\boldsymbol{\theta} + \alpha \boldsymbol{v}}), \boldsymbol{y}^{(i)}\right) \right], \quad \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}.$$

v: velocity or momentum with unit mass ($\mathbf{p} = m\mathbf{v} = 1 \cdot \mathbf{v}$)

- *Provides a direction and speed at which the parameters move through parameter space*



Accelerated gradient

$\alpha \in [0,1]$: a hyperparameter about the momentum

- *Determines how quickly the contributions of previous gradients exponentially decay.*

AncoraSIR.com

Algorithms with Adaptive Learning Rates

Incremental (or mini-batch-based) methods that adapt the learning rates of model parameters.

Algorithm 8.4 The AdaGrad algorithm

Require: Global learning rate ϵ
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant δ, perhaps 10^{-7}, for numerical stability
  Initialize gradient accumulation variable $\boldsymbol{r} = \boldsymbol{0}$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Accumulate squared gradient: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$
    Compute update: $\Delta\boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$. (Division and square root applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
  end while

Adapts the learning rates of all model parameters

- Scaling them inversely proportional to the square root of the sum of all of their historical squared values

For training deep neural network models—the accumulation of squared gradients *from the beginning of training* can result in a premature and excessive decrease in the effective learning rate.

AncoraSIR.com

Algorithms with Adaptive Learning Rates

Root Mean Squared Propagation

Root Mean Squared Propagation

Algorithm 8.5 The RMSProp algorithm

Require: Global learning rate ϵ, $\boxed{\text{decay rate } \rho.}$
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant δ, usually 10^{-6}, used to stabilize division by small numbers.
  $\boxed{\text{Initialize accumulation variables } \boldsymbol{r} = 0}$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    $\boxed{\text{Accumulate squared gradient: } \boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1-\rho)\boldsymbol{g} \odot \boldsymbol{g}}$
    Compute parameter update: $\Delta\boldsymbol{\theta} = -\frac{\epsilon}{\sqrt{\delta + \boldsymbol{r}}} \odot \boldsymbol{g}$. ($\frac{1}{\sqrt{\delta + \boldsymbol{r}}}$ applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
  end while

In AdaGrad: Accumulate squared gradient: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$

Apply Exponentially Weighted Averages

$$v_t = \beta v_{t-1} + (1-\beta)\theta_t$$

RMSProp with Nesterov Momentum

Algorithm 8.6 RMSProp algorithm with Nesterov momentum

Require: Global learning rate ϵ, decay rate ρ, $\boxed{\text{momentum coefficient } \alpha.}$
Require: Initial parameter $\boldsymbol{\theta}$, $\boxed{\text{initial velocity } \boldsymbol{v}.}$
  Initialize accumulation variable $\boldsymbol{r} = \boldsymbol{0}$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    $\boxed{\text{Compute interim update: } \tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}}$
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$
    Accumulate gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1-\rho)\boldsymbol{g} \odot \boldsymbol{g}$
    $\boxed{\text{Compute velocity update: } \boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \frac{\epsilon}{\sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}.}$ ($\frac{1}{\sqrt{\boldsymbol{r}}}$ applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
  end while

It allows for individual adjustment of the learning rate for each parameter of the model

Algorithms with Adaptive Learning Rates

Adaptive Moment (Adam) Estimation

Momentum is incorporated directly as an estimate of the first order moment (with exponential weighting) of the gradient.

Adam includes bias corrections to the estimates of both the first-order moments (the momentum term) and the (uncentered) second-order moments to account for their initialization at the origin.

Adam is generally regarded as being *fairly robust to the choice of hyperparameters*,

- Though the learning rate sometimes needs to be changed from the suggested default.

Algorithm 8.7 The Adam algorithm

Require: Step size ϵ (Suggested default: 0.001)
Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$. (Suggested defaults: 0.9 and 0.999 respectively)
Require: Small constant δ used for numerical stabilization. (Suggested default: 10^{-8})
Require: Initial parameters $\boldsymbol{\theta}$
  $\boxed{\text{Initialize 1st and 2nd moment variables } \boldsymbol{s} = \boldsymbol{0}, \boldsymbol{r} = \boldsymbol{0}}$
  Initialize time step $t = 0$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    $t \leftarrow t + 1$
    Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1) \boldsymbol{g}$
    Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2) \boldsymbol{g} \odot \boldsymbol{g}$
    Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1 - \rho_1^t}$
    Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \rho_2^t}$
    Compute update: $\Delta \boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta}$ (operations applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$
  end while

AncoraSIR.com

Choosing the Right Optimization Algorithm

Unfortunately, no consensus on this point now

Algorithms with ***adaptive learning rates*** performed fairly robustly,

- But no single best algorithm has emerged

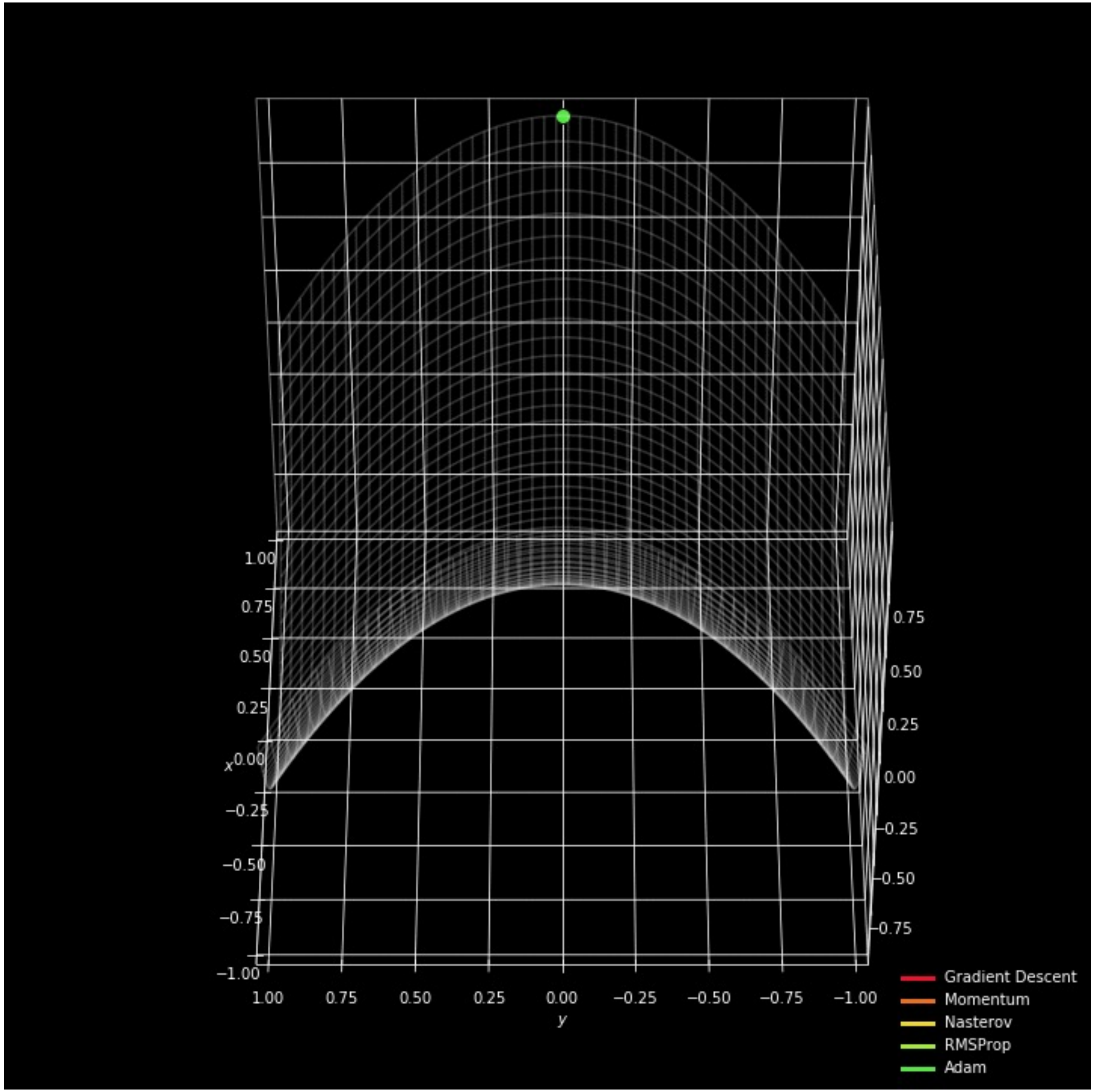
Currently, the most popular optimization algorithms actively in use

- SGD
- SGD with momentum
- RMSProp
- RMSProp with momentum
- AdaDelta
- Adam

The choice depends largely on the user's familiarity with the algorithm

- For ease of hyperparameter tuning

AncoraSIR.com

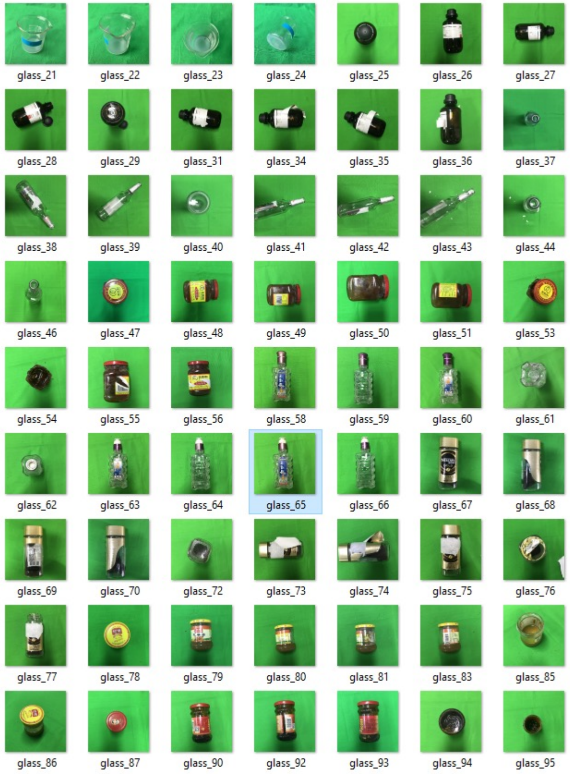


Train a Network of Trash Sorting

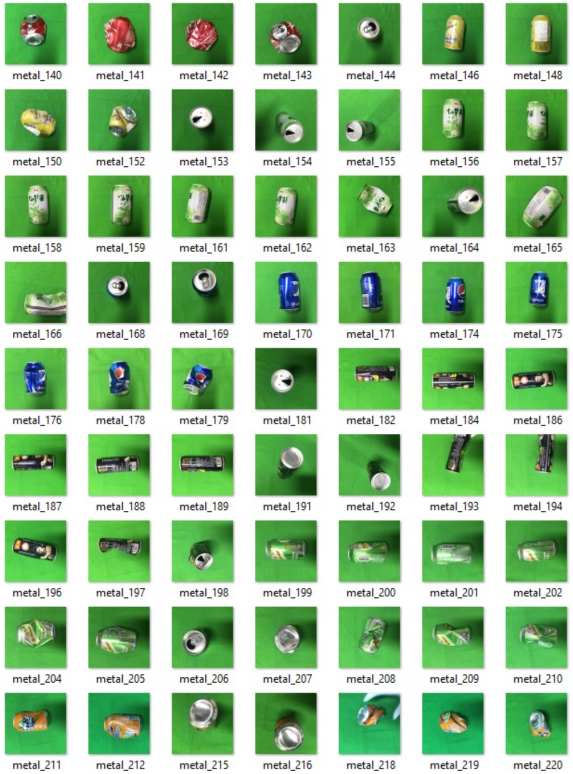
Recyclable Waste Sorting Dataset

- Four classes of waste for sorting

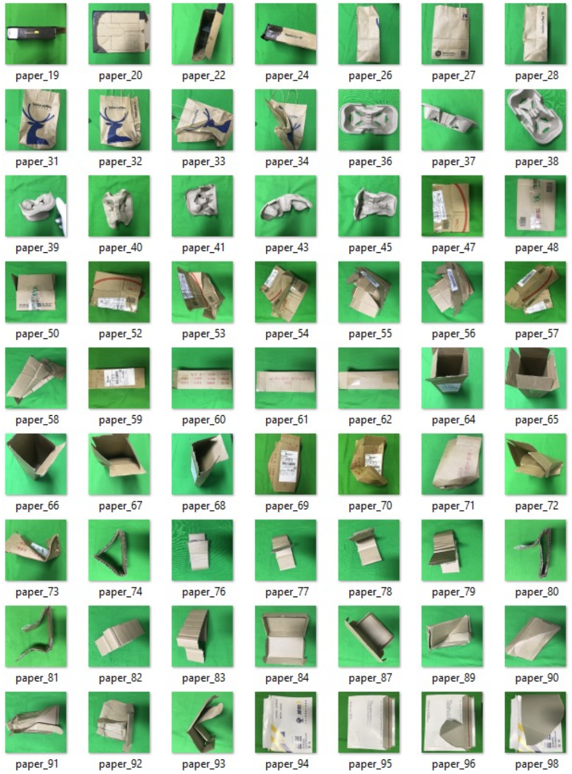
Glass



Metal



Paper



Plastic

AncoraSIR.com

SUSTech
Southern University of Science and Technology

Download Dataset

- Download the dataset from website:
  - https://pan.baidu.com/s/1IQuPEPhxA6N97AZrJU4Hmw
  - with code: 69cs

- Extract the files to have the following folder structure:
  - BionicDL
    - _train_: 80% of the total data
      - glass, metal, paper, plastic
    - _test_: 20% of the total data
      - glass, metal, paper, plastic

	Train (80%)	Test (20%)	Total
Glass	204	50	254
Metal	396	98	494
Paper	1291	322	1613
Plastic	1202	300	1502
Total	3093	770	3863

AncoraSIR.com

Data Generator & Augmentation

With TensorFlow and Keras

- The data generator class in *tensorfow.keras* can conveniently generate data flow and add data augmentation for GPU training.

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

BATCH_SIZE = 32

# Data generators, data augmentation
# You can uncomment the data augmentation parameters
train = ImageDataGenerator(
    samplewise_center=True,
#     rotation_range = 90.0,
#     width_shift_range=0.2,
#     height_shift_range=0.2,
#     horizontal_flip = True,
    vertical_flip = False)

valid = ImageDataGenerator(samplewise_center=True)
```

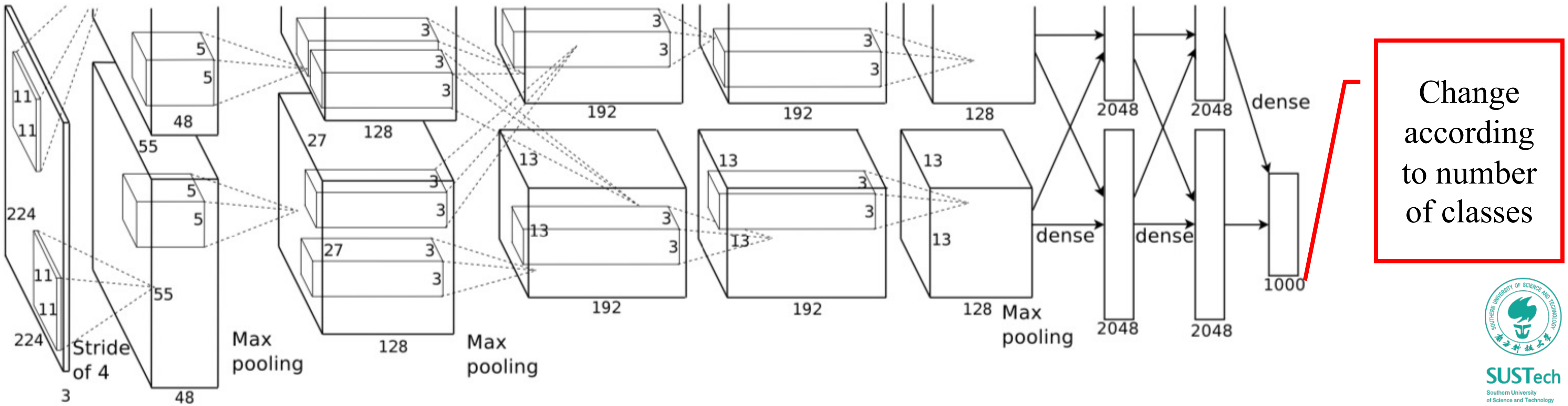
```
# Target directories
trainGenerator = train.flow_from_directory(
    '../data/BionicDL/_train_',
    target_size = (224, 224),
    batch_size=BATCH_SIZE,
    class_mode = "categorical",
    color_mode = "rgb", shuffle=True, seed=42)

validationGenerator = valid.flow_from_directory(
    '../data/BionicDL/_test_',
    target_size = (224, 224),
    batch_size=1,
    class_mode = "categorical",
    color_mode = "rgb", shuffle=False)
```

```
Found 3093 images belonging to 4 classes.
Found 770 images belonging to 4 classes.
```

AlexNet Model

- **Architecture**: five convolutional layers and three fully-connected layers.
- **Contributions**:
  - ReLU instead of Tanh to add non-linearity. It accelerates the speed by 6 times at the same accuracy.
  - Dropout instead of regularization to deal with overfitting
  - Overlay pooling to reduce the size of network



AncoraSIR.

Build the Model in TensorFlow

```python
model = Sequential(name="AlexNet")

# 1st Convolutional Layer
model.add(Conv2D(filters=96, input_shape=(224,224,3), kernel_size=(11,11),strides=(4,4), padding='valid',
name='conv_1'))
model.add(Activation('relu'))
# Pooling
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='valid'))
# Batch Normalisation before passing it to the next layer
model.add(BatchNormalization())

# 2nd Convolutional Layer
model.add(Conv2D(filters=256, kernel_size=(11,11), strides=(1,1), padding='valid', name='conv_2'))
model.add(Activation('relu'))
# Pooling
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='valid'))
# Batch Normalisation
model.add(BatchNormalization())

# 3rd Convolutional Layer
model.add(Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), padding='valid', name='conv_3'))
model.add(Activation('relu'))
# Batch Normalisation
model.add(BatchNormalization())

# 4th Convolutional Layer
model.add(Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), padding='valid', name='conv_4'))
model.add(Activation('relu'))
# Batch Normalisation
model.add(BatchNormalization())

# 5th Convolutional Layer
model.add(Conv2D(filters=256, kernel_size=(3,3), strides=(1,1), padding='valid', name='conv_5'))
model.add(Activation('relu'))
# Pooling
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='valid'))
# Batch Normalisation
model.add(BatchNormalization())
```

```python
# Passing it to a dense layer
model.add(Flatten())
# 1st Dense Layer
model.add(Dense(4096, input_shape=(224*224*3,), name='dense_1'))
model.add(Activation('relu'))
# Add Dropout to prevent overfitting
model.add(Dropout(0.4))
# Batch Normalisation
model.add(BatchNormalization())

# 2nd Dense Layer
model.add(Dense(4096, name='dense_2'))
model.add(Activation('relu'))
# Add Dropout
model.add(Dropout(0.4))
# Batch Normalisation
model.add(BatchNormalization())

# Output Layer
model.add(Dense(4, name='dense_3_new'))
model.add(Activation('softmax'))
```

Model Summary

24 million parameters with Default random weights initialization

```
Model: "AlexNet"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv_1 (Conv2D)              (None, 54, 54, 96)        34944
_________________________________________________________________
activation (Activation)      (None, 54, 54, 96)        0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 27, 27, 96)        0
_________________________________________________________________
batch_normalization (BatchNo (None, 27, 27, 96)        384
_________________________________________________________________
conv_2 (Conv2D)              (None, 17, 17, 256)       2973952
_________________________________________________________________
activation_1 (Activation)    (None, 17, 17, 256)       0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 256)         0
_________________________________________________________________
batch_normalization_1 (Batch (None, 8, 8, 256)         1024
_________________________________________________________________
conv_3 (Conv2D)              (None, 6, 6, 384)         885120
_________________________________________________________________
activation_2 (Activation)    (None, 6, 6, 384)         0
_________________________________________________________________
batch_normalization_2 (Batch (None, 6, 6, 384)         1536
_________________________________________________________________
conv_4 (Conv2D)              (None, 4, 4, 384)         1327488
_________________________________________________________________
activation_3 (Activation)    (None, 4, 4, 384)         0
_________________________________________________________________
batch_normalization_3 (Batch (None, 4, 4, 384)         1536
_________________________________________________________________
```


```
conv_4 (Conv2D)              (None, 4, 4, 384)         1327488
_________________________________________________________________
activation_3 (Activation)    (None, 4, 4, 384)         0
_________________________________________________________________
batch_normalization_3 (Batch (None, 4, 4, 384)         1536
_________________________________________________________________
```


```
conv_5 (Conv2D)              (None, 2, 2, 256)         884992
_________________________________________________________________
activation_4 (Activation)    (None, 2, 2, 256)         0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 1, 1, 256)         0
_________________________________________________________________
batch_normalization_4 (Batch (None, 1, 1, 256)         1024
_________________________________________________________________
flatten (Flatten)            (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 4096)              1052672
_________________________________________________________________
activation_5 (Activation)    (None, 4096)              0
_________________________________________________________________
dropout (Dropout)            (None, 4096)              0
_________________________________________________________________
batch_normalization_5 (Batch (None, 4096)              16384
_________________________________________________________________
dense_2 (Dense)              (None, 4096)              16781312
_________________________________________________________________
activation_6 (Activation)    (None, 4096)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 4096)              0
_________________________________________________________________
batch_normalization_6 (Batch (None, 4096)              16384
_________________________________________________________________
dense_3_new (Dense)          (None, 4)                 16388
_________________________________________________________________
activation_7 (Activation)    (None, 4)                 0
=================================================================
Total params: 23,995,140
Trainable params: 23,976,004
Non-trainable params: 19,136
_________________________________________________________________
```

AncoraSIR.com

Training the Model

- Training options
  - Epochs:

```
# compute steps per epoch
EPOCHS = 30
steps_per_epoch = 3093//BATCH_SIZE  # 3093 items in the training dataset
```

  - Optimizer and learning rate: SGD:

```
# you can test different optimizer settings here
# opt = tf.keras.optimizers.SGD(lr=0.01)
opt = tf.keras.optimizers.SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
```

  - Callbacks: functions to be executed during the training process:

```
# utility callback that displays training curves
plot_training = PlotTraining(sample_rate=1, zoom=1)
checkpoint = tf.keras.callbacks.ModelCheckpoint(
    'BionicDL-bs32-weights.{epoch:02d}-{val_acc:.3f}-DenseNet169.hdf5',
    verbose=1, save_best_only=True,
    save_weights_only=False,
    mode='max', period=1)
```

  - Starting training:

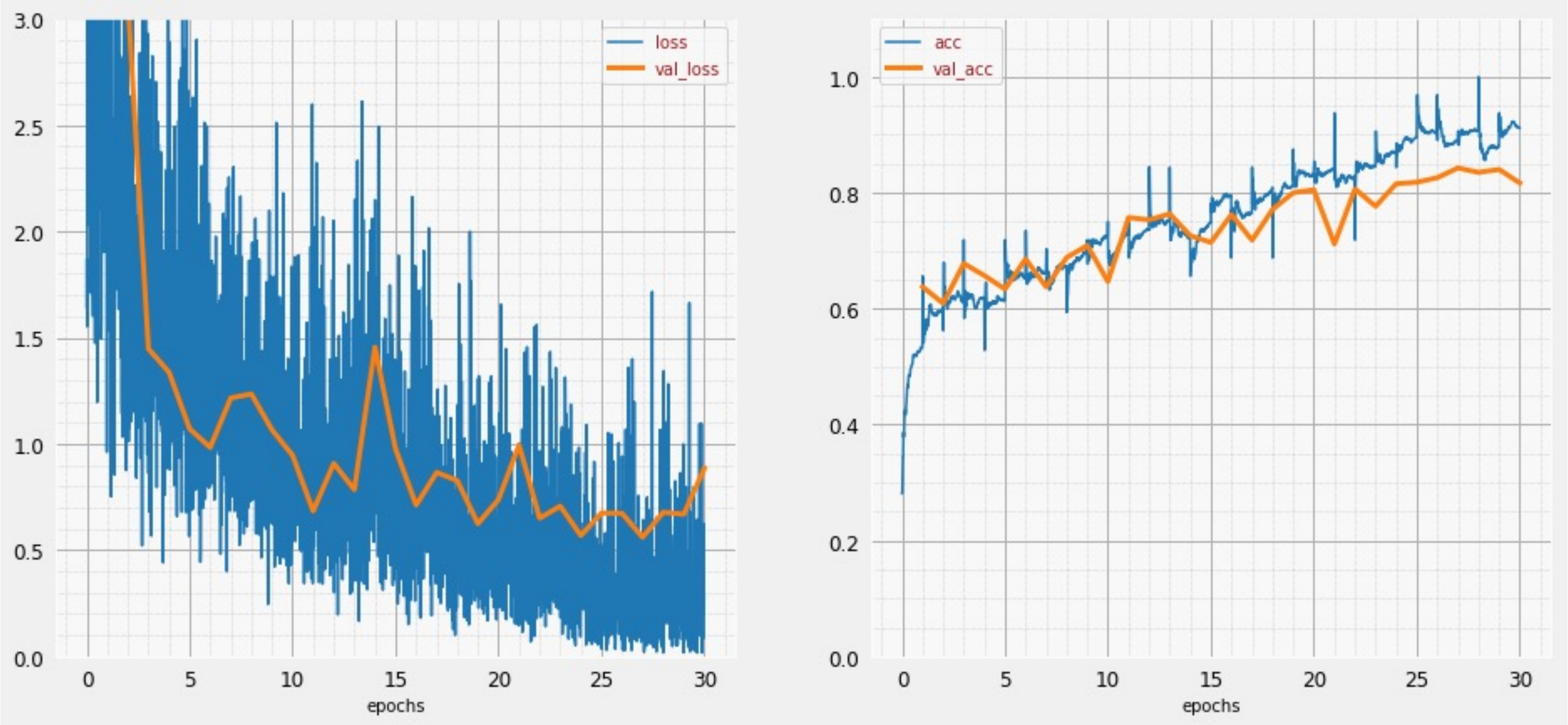
```
# Compile
model.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy'])

history = model.fit_generator(trainGenerator,
                              steps_per_epoch=steps_per_epoch,
                              epochs=EPOCHS,
                              validation_data=validationGenerator,
                              validation_steps=770,
                              callbacks=[plot_training, checkpoint])
```

AncoraSIR.com

Training the Model

- Training VS Validation:
  - Validation accuracy is saturated at 85% while training accuracy is approaching 100%.

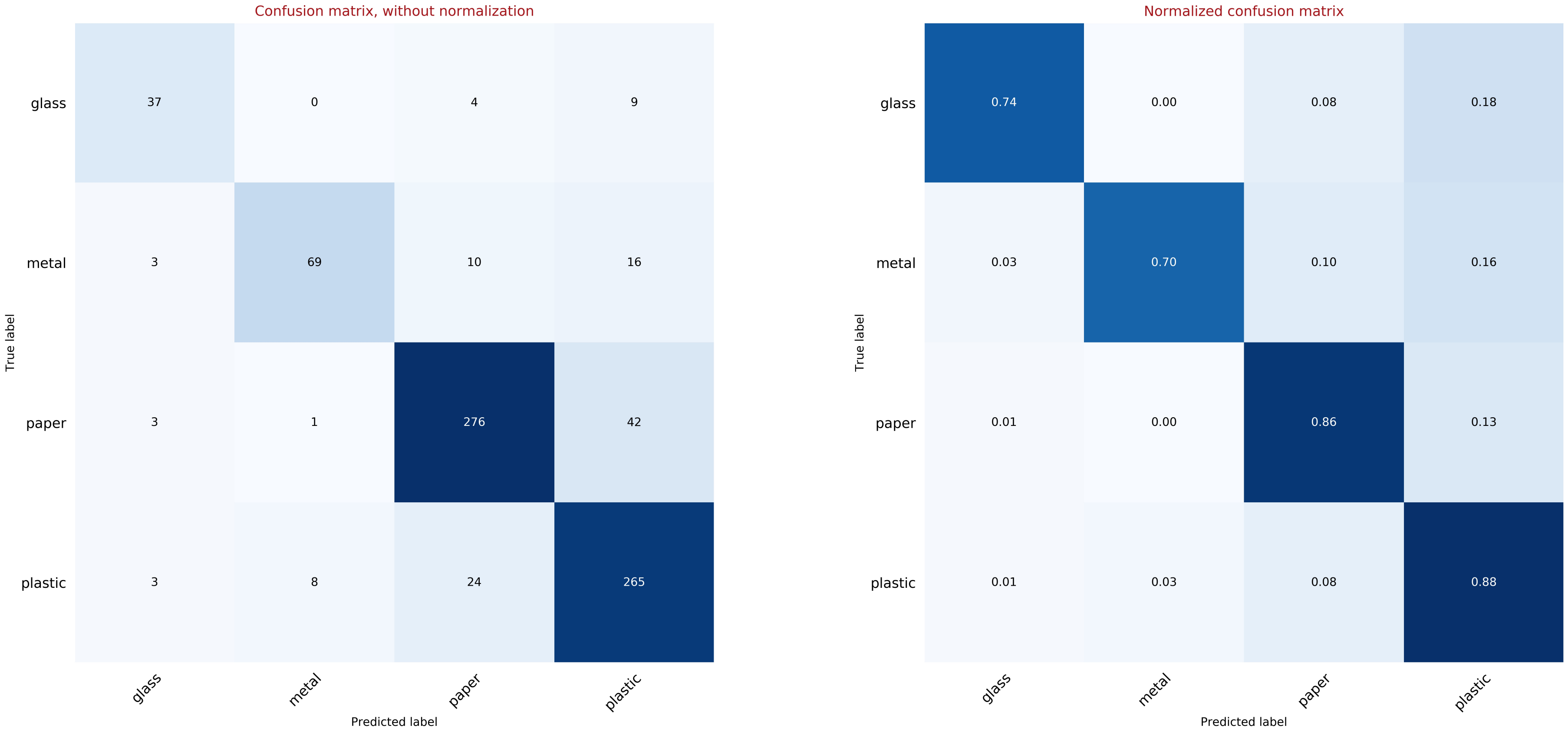


AncoraSIR.com

Results

Check the prediction accuracy of each category

- Confusion matrix



AncoraSIR.com

Thank you~

songcy@sustech.edu.cn