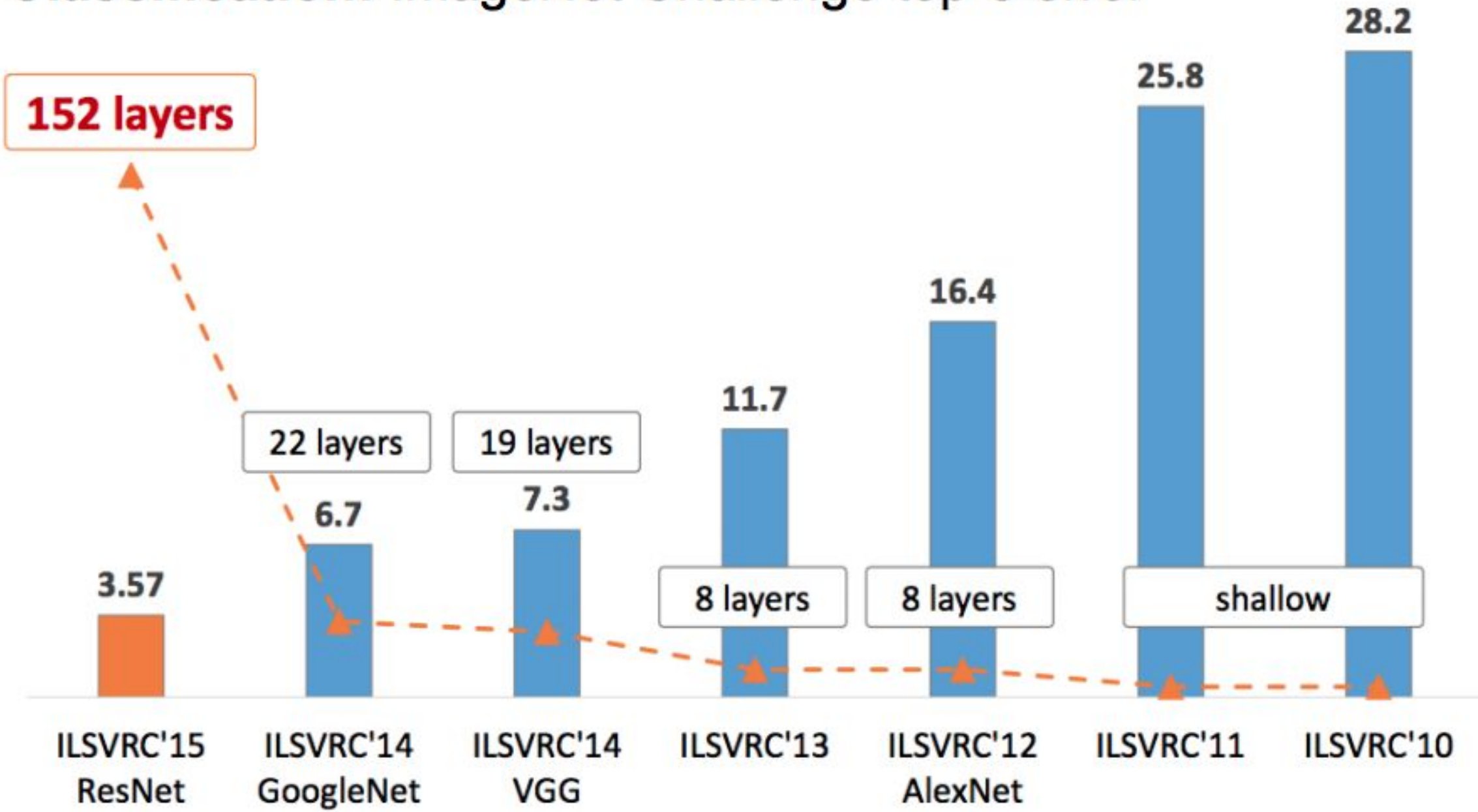
Day 6

CNN/RNN

CNN Architectures

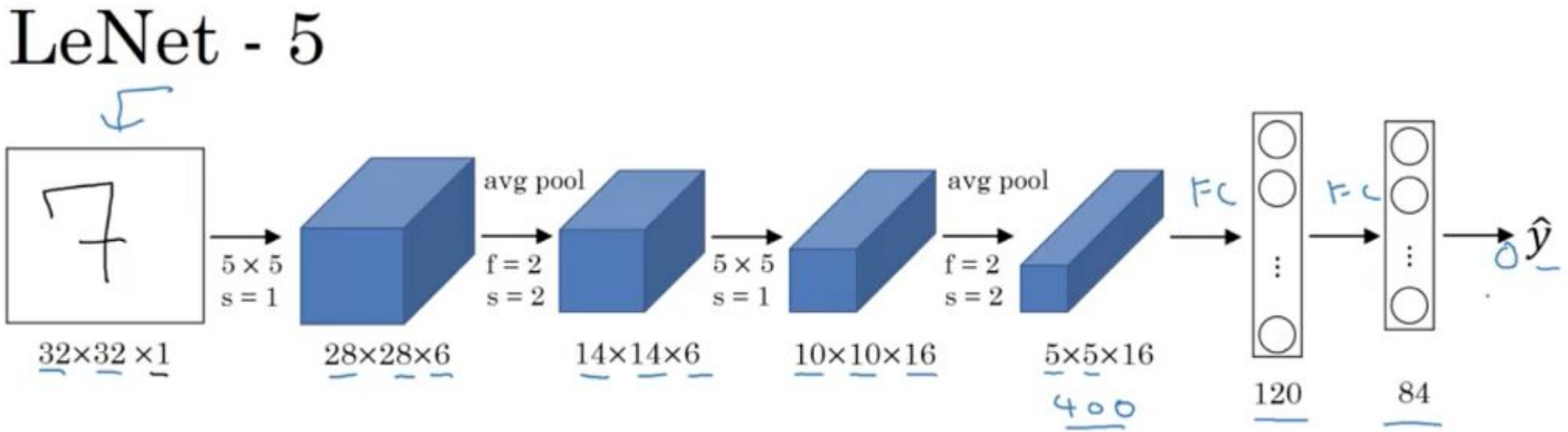
- Popular CNN architectures won in ILSVRC (ImageNet Large Scale Vision Recognition Challenge) competitions
  - LeNet-5
  - AlexNet
  - VGGNet
  - GoogLeNet
  - ResNet
- Participants are provided with 1.4 millions of images

Classification: ImageNet Challenge top-5 error



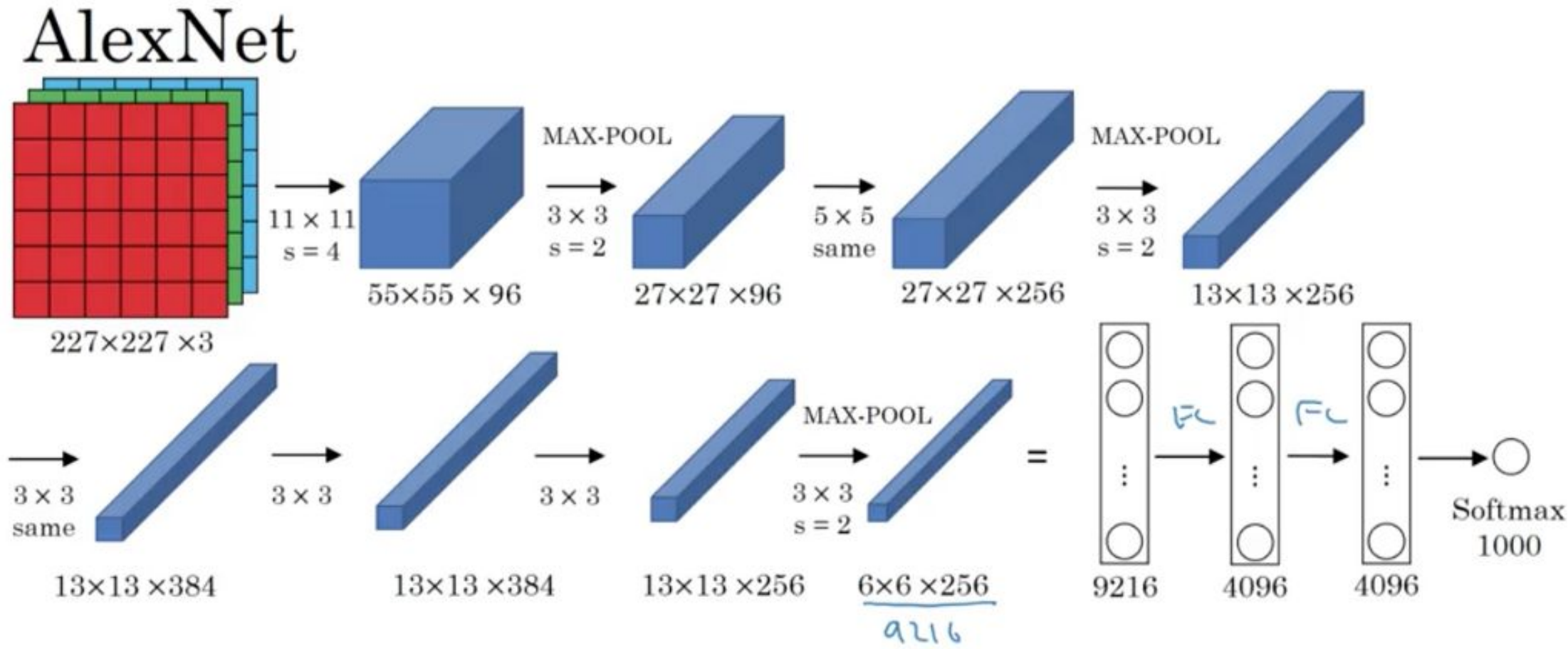
LeNet-5

- Classical neural network architecture successfully used on MNIST handwritten digit recognizer patterns



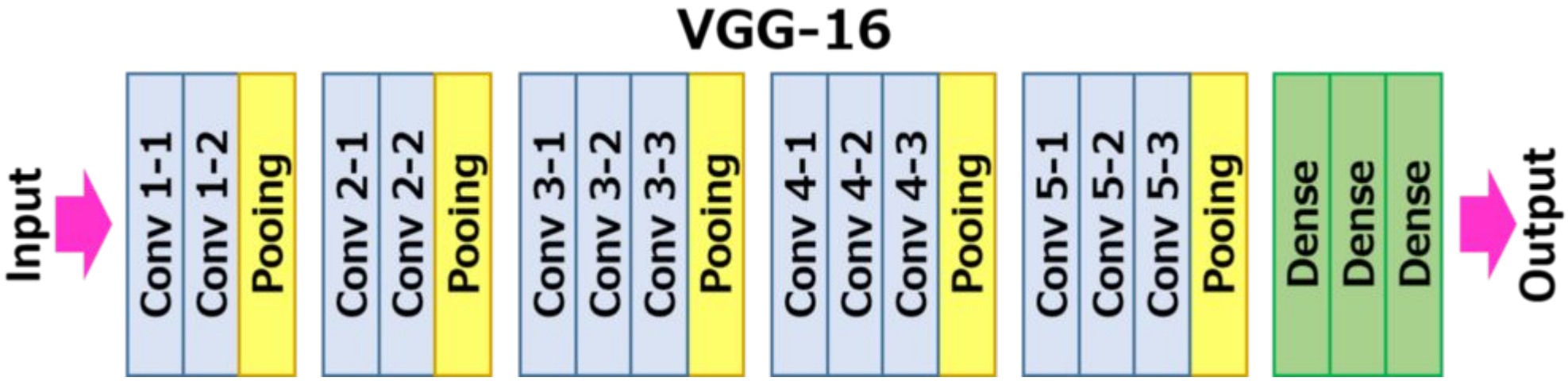
AlexNet

Total 60 million parameters



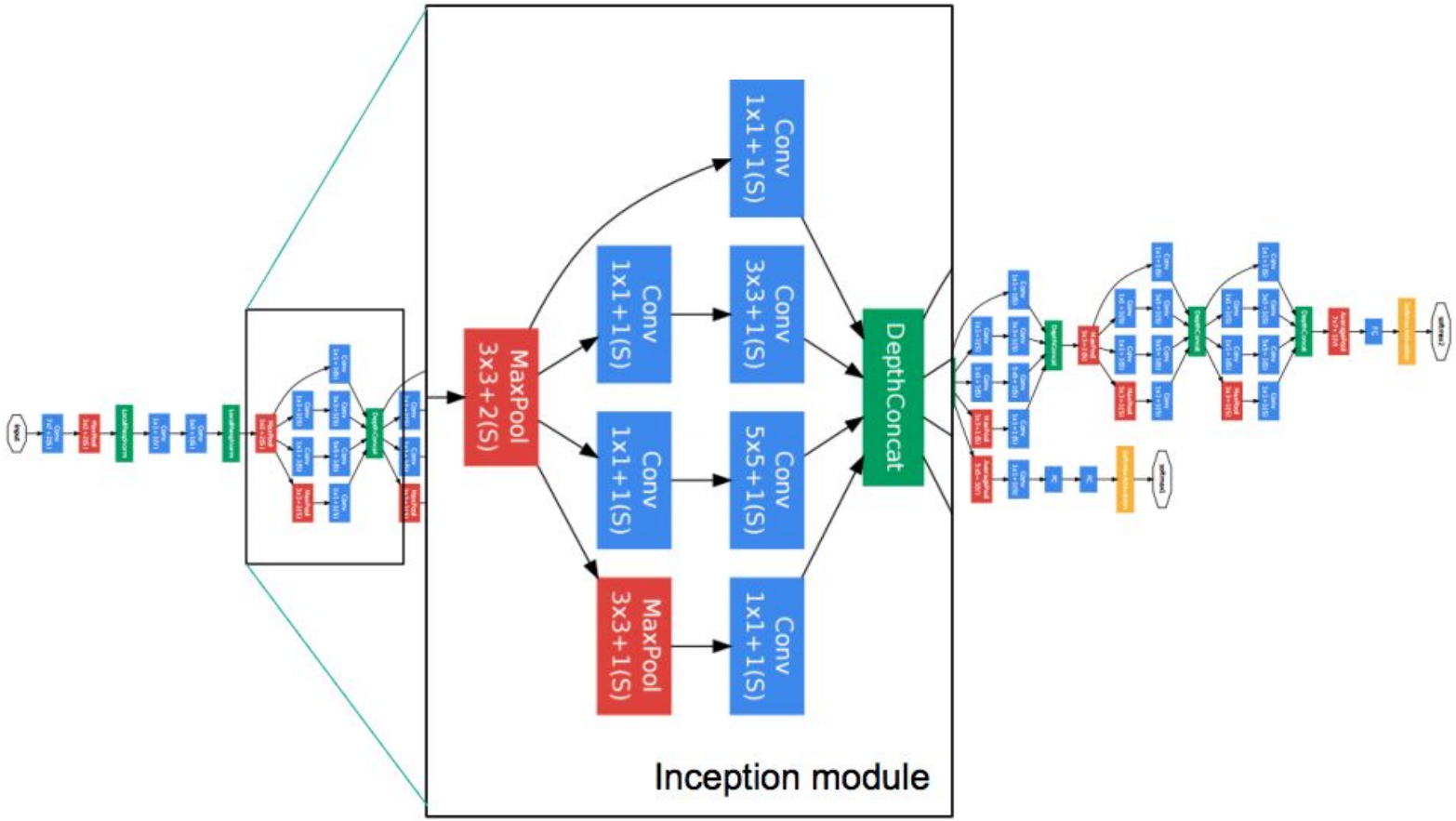
VGG-16

- A simpler architecture model with less hyper parameters



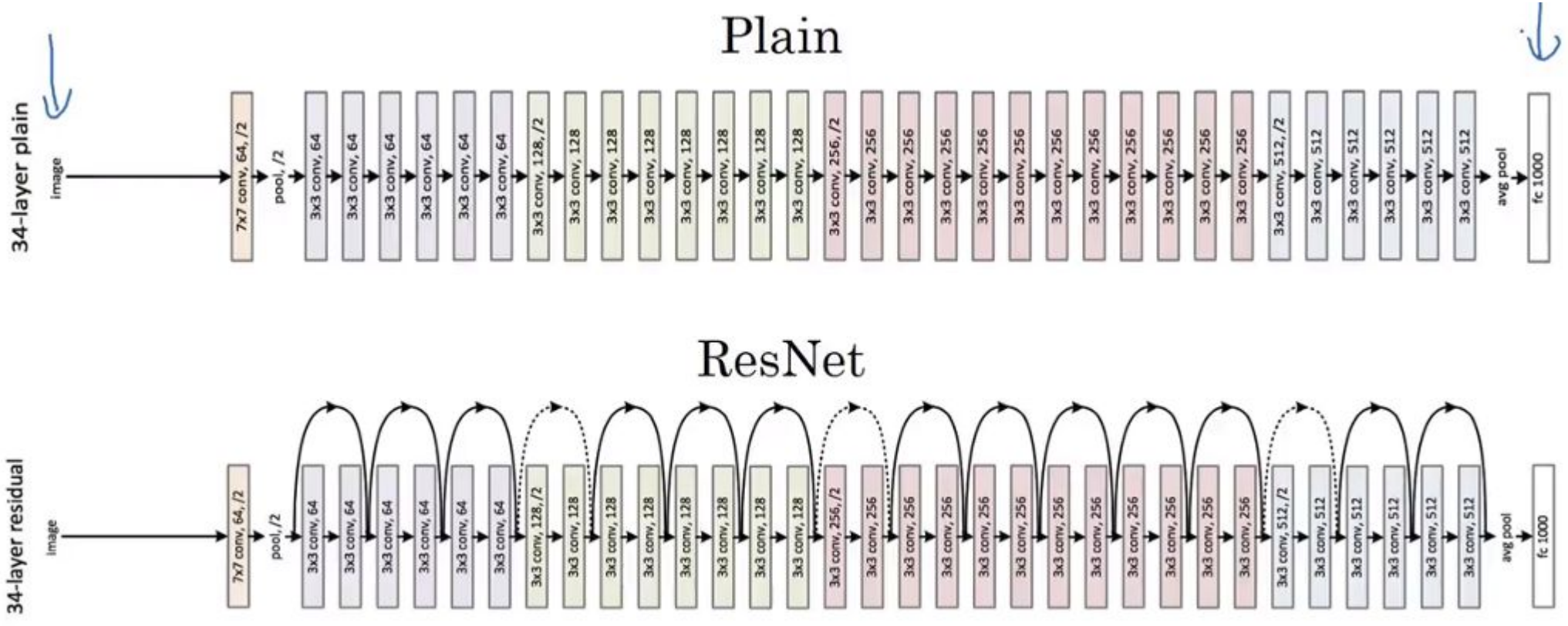
GoogLeNet

- Also known as Inception Module



ResNet (Residual Neural Network)

- Introduced a concept called “skip connections



ReLU vs Leaky ReLU

- RelU (Rectified Linear Unit)
  - Become very popular in the last few years. But, ReLU units can die during training. For example, about 40% of the network never activate across the entire training dataset
  - You can avoid this issue with a proper setting of the learning rate (smaller)
- Leaky ReLU
  - When $x < 0$, have a small negative slope of 0.01 or so

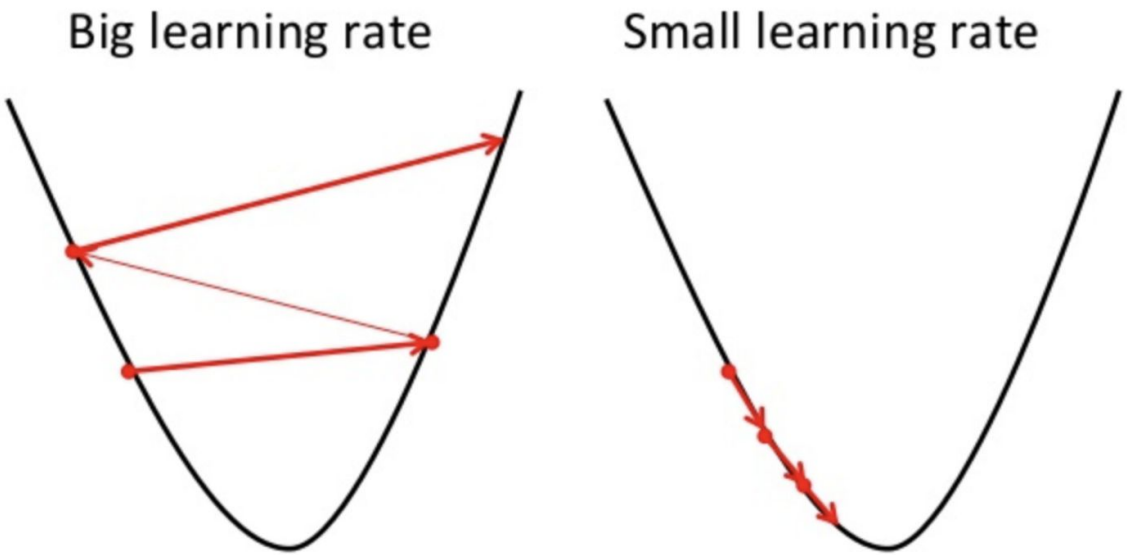
Adam (Adaptive Moment Estimation)

- Another method that computes the adaptive learning rates for each parameter by considering the exponentially decaying average of past squared gradients and the exponentially decaying average of past gradients. **기하급수적으로 쇠퇴하는 과거의 경사도의 제곱의 평균과 기하급수적으로 쇠퇴하는 과거경사도의 평균을 고려해서 각 계수에 대해 학습률을 계산**

```
opt = tf.keras.optimizers.Adam(lr=1e-3, decay=13-5)

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

learning rate and decay

- Learning rate is the amount that the weights are updated during training
  트레이닝할때 가중치가 업데이트되는 정도
- For example, start with 0.1, then try exponentially lower values: 0.01, 0.001, etc.
- When making the weight update, penalizing large weights. **가중치를 업데이트할때 너무 큰 값은 패널라이즈 (decay)**



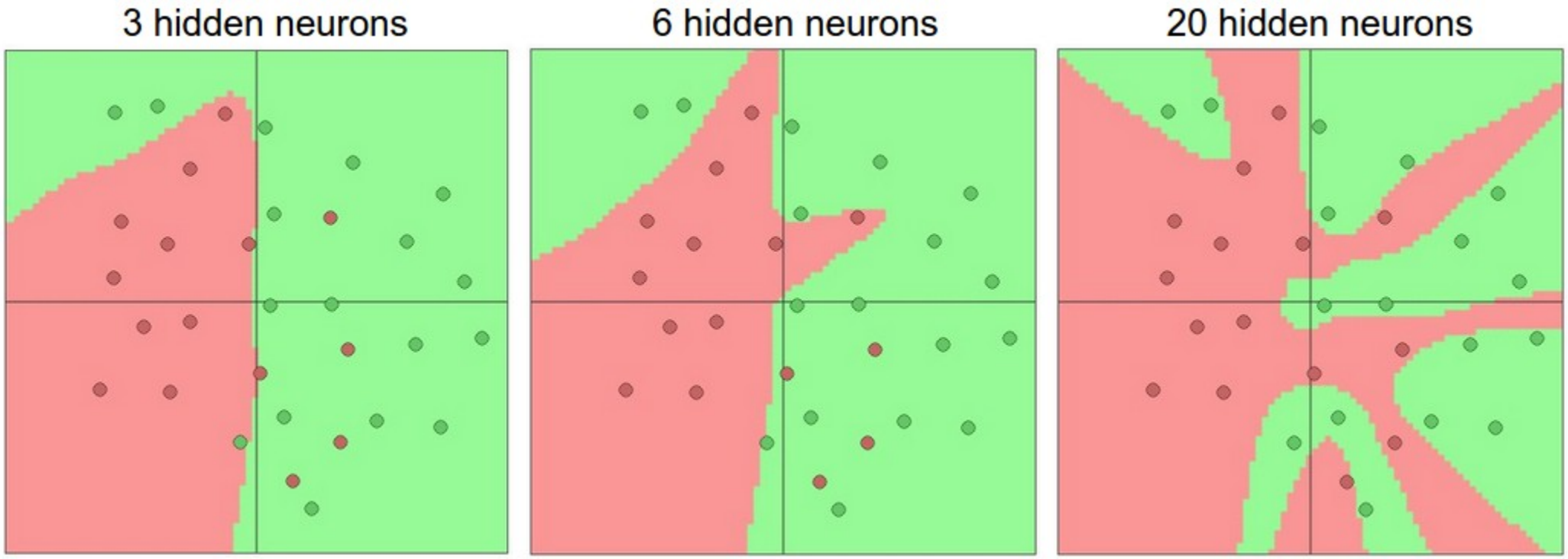
ADAgrad Optimizer

- Uses a different learning rate for every parameter and every time step
  계수마다 시간간격마다 학습률이 다름
- The parameters that are infrequent must have larger learning rates while parameters that are frequent must have smaller learning rates. That is, the stochastic gradient descent update for parameters 자주 쓰이지 않는 계수는 학습률이 크고, 자주 쓰이는 계수는 학습률을 낮게 조정. 확률적인 계수의 업데이트를 사용
- The learning rate is calculated based on the past gradients that have been computed for each parameter. 계수 계산시 사용됐던 경사도가 학습률 계산에 사용됨
- The learning rates start vanishing very quickly as the iterations increase.
  반복될수록 학습률의 빨리 사라짐

RMSprop

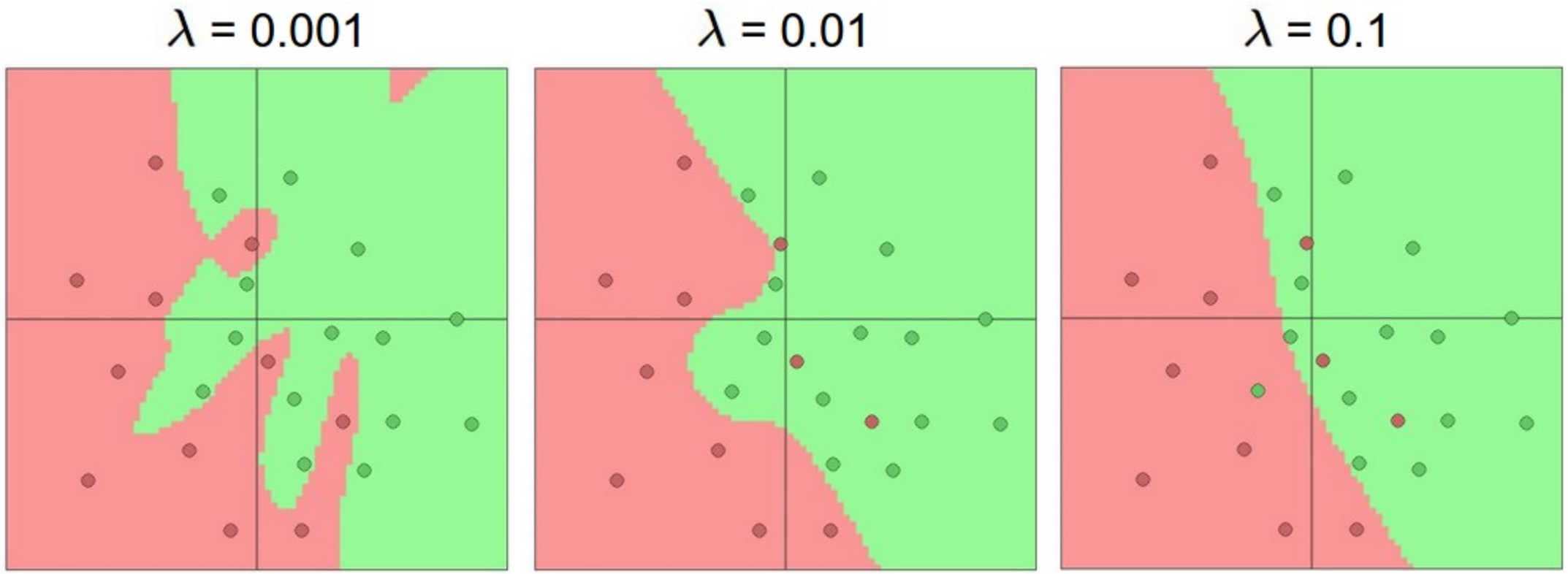
- Considers fixing the diminishing learning rate by only using a certain number of previous gradients.
  과거 경사도중 특정한 갯수를 사용해서 학습률이 줄어드는 것을 고정시킴

Number of Hidden Layers and Size



With one hidden layer and different sizes

Regularization



With 20 hidden neurons, the final decision regions is smoother with a higher regularization 정규화를 시킬수록 구분되는 영역이 더 스무드함

Labeled Faces in the Wild

- A dataset of face photographs designed for studying the problem of unconstrained face recognition. It contains more than 13,000 images of faces collected from the web. Each face has been labeled with the name of the person pictured. 1680 of the people pictured have two or more distinct photos in the data set.
  얼굴인식을 위해 얼굴사진을 웹에서 모아논 데이터셋. 13,000개의 이미지를 가지고 있음. 각 얼굴사진은 이름과 함께 저장되어 있고, 그중 1680명의 사람은 각각 다른 사진이 2개이상의 들어있음

http://vis-www.cs.umass.edu/lfw/#download

Image Data Input Parameters

- number of images = 100
- image width, image height =100 pixels
- 3 channels, pixel levels in the range [0–255]

Exercise #6

- Open 100 images from Labeled Faces in the Wild 100
  100개의 이미지를 오픈
  - Resize the image to 100x100 이미지의 사이즈를 바꾸기
  - Rotate 130 degree **130도 회전**
  - Gray scale 그레이스케일
  - Convert images to numpy array 넘파이 배열로 바꾸기
  - Mean and standard deviation of images 이미지의 평균, 표준편차
  - Normalize images 정규화
- Convert to 4d array **4D 배열로 바꾸기**

Handling Images in Many Folders

- Make the required folders(validation and the class folders). You can get this done inside the script(os.makedirs). **훈련, 테스트폴더 만들기**
- Get the number of images in the ‘train’ folder.(len(os.listdirs())) **훈련폴더에 있는 전체 이미지를 넣고 갯수를 셈**
- Copy 20 percent(as much as you want) images randomly chosen to the validation class folders. (random.choice(os.listdir()) and shutil.move()) **20%의 이미지들을 테스트폴더로 옮김**

College Dataset

Use College Data Set which has several features of a college and a categorical column and build a predictive model to determine whether or not the School is Public or Private

컬리지 데이터셋에는 대학에 관한 여러가지 특징들과 범주형 변수들이 포함돼 있습니다. 그 대학이 공립인지 사립인지 결정하는 예측모델을 구축하시요.

Variables

- Private : Public/private indicator
- Apps : Number of applications received
- Accept : Number of applicants accepted
- Enroll : Number of new students enrolled
- Top10perc : New students from top 10% of high school class
- Top25perc : New students from top 25% of high school class
- F.Undergrad : Number of full-time undergraduates
- P.Undergrad : Number of part-time undergraduates
- Outstate : Out-of-state tuition
- Room.Board : Room and board costs
- Books : Estimated book costs
- Personal : Estimated personal spending
- PhD : Percent of faculty with Ph.D.'s
- Terminal : Percent of faculty with terminal degree
- S.F.Ratio : Student/faculty ratio
- perc.alumni : Percent of alumni who donate
- Expend : Instructional expenditure per student
- Grad.Rate : Graduation rate

It contains a number of variables for 777 different universities and colleges in the US.

Exercise #6

- Load College data (pd.read_csv)
- Data exploration (head, describe, isna.sum(), set_index)
- X and y split
- Data preprocessing (y - LabelEncoder, x - StandardScaler)
- Train and test split (sklearn.model_selection.train_test_split)

Exercise #6

- Build a CNN Model (First layer: 12 nodes, with relu activation, 2nd layer: 8 nodes, with relu activation, 3rd layer: output, with sigmoid activation)
  - Use init='uniform‘ for weight and bias initializers
- Compile the model (loss='binary_crossentropy',optimizer='adam', metrics=['accuracy'])
- Fit the model (epochs=1000, batch_size=16)
- Prediction and evaluation (model.predict_classes)
- Accuracy and loss chart

ModelCheckpoint

- You can setup to save the network weights only when there is an improvement in classification accuracy on the validation dataset
  테스트 데이터셋의 정확도가 높아질때만 가중치를 저장할수 있음

Checkpoint Code

```
# checkpoint

ckpt_model="weights.best.hdf5"

checkpoint = ModelCheckpoint(ckpt_model, monitor='val_accuracy',
verbose=1, save_best_only=True, mode='max')

callbacks_list = [checkpoint]


# Fit the model

model.fit(X, Y, validation_split=0.33, epochs=150, batch_size=10,
callbacks=callbacks_list, verbose=0)
```

Loading Checkpoint

```
# create model
…
# load weights
model.load_weights("weights.best.hdf5")
# compile model (이미 fit된 상태라 fit할 필요없음)
…
```

Boston Housing Dataset

The Boston dataset is a collection of data about housing values in the suburbs of Boston. Use the Boston dataset in the MASS package and predict the median value of owner-occupied homes (medv) using all the other continuous variables available.

보스톤 데이터셋에는 보스톤 교외에 있는 집의 가치에 대한 데이터를 포함합니다. 포함되어 있는 모든 연속변수를 사용하여 집의 중위값을 예측하시요.

Variables

- CRIM - per capita crime rate by town
- ZN - proportion of residential land zoned for lots over 25,000 sq.ft.
- INDUS - proportion of non-retail business acres per town.
- CHAS - Charles River dummy variable (1 if tract bounds river; 0 otherwise)
- NOX - nitric oxides concentration (parts per 10 million)
- RM - average number of rooms per dwelling
- AGE - proportion of owner-occupied units built prior to 1940
- DIS - weighted distances to five Boston employment centres
- RAD - index of accessibility to radial highways
- TAX - full-value property-tax rate per $10,000
- PTRATIO - pupil-teacher ratio by town
- B - 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
- LSTAT - % lower status of the population
- MEDV - Median value of owner-occupied homes in $1000's

Exercise #6

- Load Boson data
  - from sklearn.datasets import load_boston
- X and y split
  - x,y = load_boston(return_X_y=True)
- Data exploration (head, describe, isna.sum())
- Data preprocessing (StandardScaler)
- Train and test split (sklearn.model_selection.train_test_split)

Loading Datasets

- https://keras.io/datasets/

```
from keras.datasets
import boston_housing

(x_train, y_train),
(x_test, y_test) =
boston_housing.load_data
()
```

- https://scikit-learn.org/0.16/modules/classes.html#module-sklearn.datasets

```
from sklearn.datasets
import load_boston

boston = load_boston()

x, y = boston.data,
boston.target
```

Exercise #6

- Build a CNN Model (First layer: 64 nodes, with relu activation, Second layer: 64 nodes, with relu activation, 3rd layer: 1 output)
  - Use kernel_initializer='normal‘
- Compile the model (loss='mean_squared_error',optimizer='adam‘, metrics=[‘mse’])
- Fit the model (epochs=100)
- Prediction and evaluation
- Accuracy and loss chart (val_mean_squrard_error)

Exercise #6

- Complete the MPG tutorial on https://www.tensorflow.org/tutorials/keras/regression
  - Preprocessing – remove null values by row (dropna)
  - Train and test split
  - Scaling x
  - Building a model (64 Dense with relu, 64 Dense with relu, 1 Dense)
  - Compile the model (loss='mse', optimizer='adam', metrics=['mae', 'mse']

Exercise #6

- Fit the model (epochs = 1000)
- Predict the mpg
- Evaluate the model
- Create History table and MAE, MSE chart

Exercise #7

- Complete the Pima-Indian Diabetes tutorial on https://www.kaggle.com/atulnet/pima-diabetes-keras-implementation
  - 난수를 고정 (np.random.seed)
  - Replace null values with average values (fillna)
  - x, y split and train, test split
  - Build a model (12 Dense with RELU, 8 Dense nodes with RELU activation, 1 Dense output with sigmoid activation)
  - Compile the model (loss='binary_crossentropy', optimizer='adam', metrics=['accuracy']

Exercise #6

- Fit the model (epochs = 1000, batch_size=16)
- Predict the diabetes
- Evaluate the model
- Create History table and accuracy and loss chart

AirPassengers

AirPassengers dataset includes monthly Airline Passenger Numbers 1949-1960. Build a prediction model for Airline Passenger Number. **에어패씬저 데이터셋에는 1949년에서 1960년까지의 월별 승객수에 대한 데이터를 포함하고 있습니다. 승객수를 예측하는 모델을 만드시요.**

Exercise #5

- Load Air Passenger data (pd.read_csv)
- Data exploration (head, describe, isna.sum(), plt.plot)
- X and y split
  - Y = number of passengers
  - X = create sequential numbers starting from 1
- Train and test split (sklearn.model_selection.train_test_split)

Exercise #5

- Build a CNN Model (64 nodes with relu, 64 nodes with relu, 1 output)
- Compile the model (loss='mean_squared_error',optimizer='adam‘, metrics=[‘mse’])
- Fit the model (epochs=400, batch_size=2)
- Prediction and evaluation
- Accuracy and loss chart (val_mean_squrard_error)
- Plot the original data and predicted data

shift()

- Used to create copies of columns that are pushed forward (rows of NaN values added to the front) or pulled back (rows of NaN values added to the end).
  데이터를 앞이나 뒤로 밀수있음

```
df = pd.DataFrame()
df['t'] = [x for x in range(10)]
df['t-1'] = df['t'].shift(1)
print(df)
```

	t	t-1
0	0	NaN
1	1	0.0
2	2	1.0
3	3	2.0
4	4	3.0
5	5	4.0
6	6	5.0
7	7	6.0
8	8	7.0
9	9	8.0

Null Values in DataFrame

```
df = pd.DataFrame()

df['t'] = [x for x in
range(10)]

df['t+1'] =
df['t'].shift(-1)

df['t+2'] =
df['t'].shift(-2)

df['t+3'] =
df['t'].shift(-3)

print(df)
```

```
df=df[1:]

df.dropna(inplace=True)
```

X and Y Split

```
x = df.iloc[:,:-1]
y = df.iloc[:, -1]
```

CNN Model

```
model = Sequential()
model.add(Dense(100, activation='relu', input_dim=3))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
```

Fit and Prediction

```
# fit model
history = model.fit(x, y, epochs=2000, verbose=0)


# demonstrate prediction
y_pred = model.predict(x)
plt.plot(y)
plt.plot(y_pred)
```

Evaluation

```
#evaluate the model
model.evaluate(x,y)
h = pd.DataFrame(history.history)
h['epoch'] = history.epoch
```

Test Data

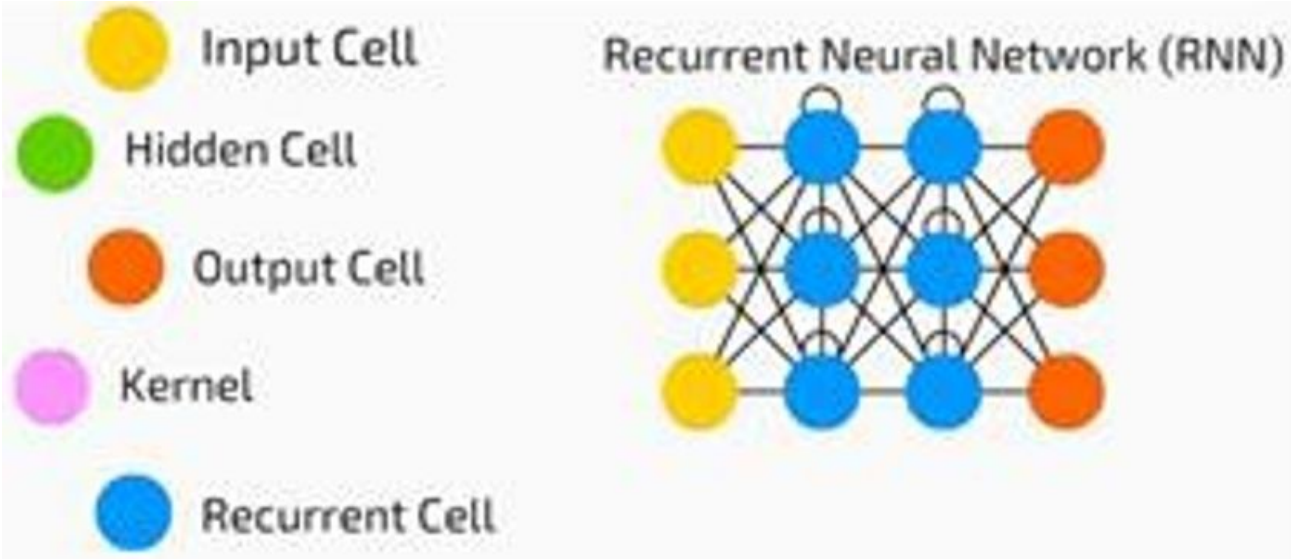
```
#test data
x_input = np.array([50, 60, 70])
x_input = x_input.reshape((1, 3))
test_pred = model.predict(x_input, verbose=0)
print(test_pred)
```

Exercise #6

- Use the shift function and build the CNN model for Air Passenger data **여객승객데이터를 이용해서 쉬프트로 이동한 후에 CNN 모델을 만드시요**

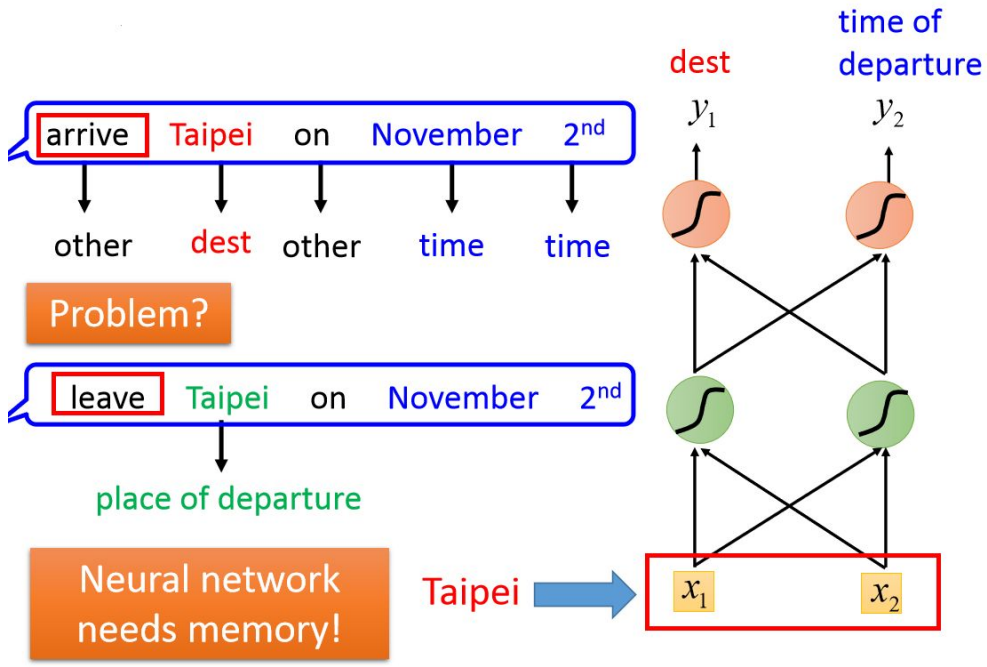
RNN (Recurrent Neural Network)
순환 인공 신경망

- A type of Neural Network where the output from previous step are fed as input to the current step
  신경망의 종류로 이전스텝의 결과물이 현재스텝의 입력값이 됨
- Language modelling or Natural Language Processing (NLP)
  자연어처리



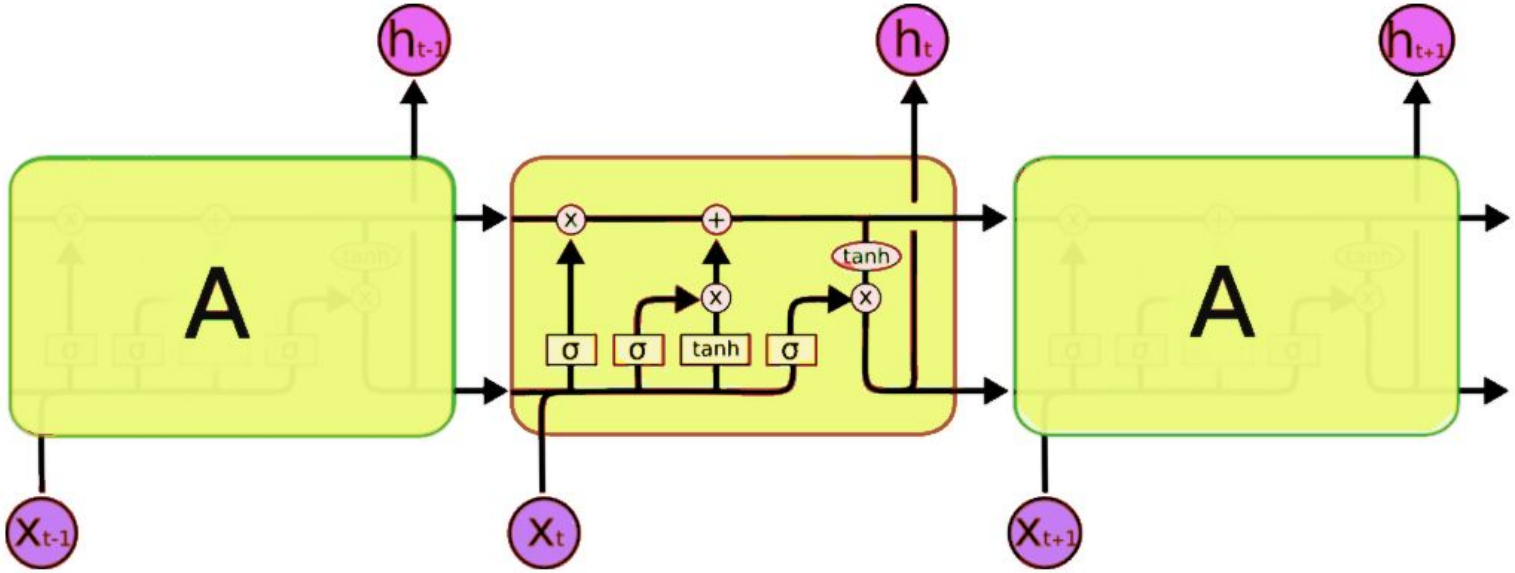
Memory

- RNN has a “memory” which captures information about what has been calculated so far.
  메모리를 이용하여 지금까지 계산한 결과를 저장해서 예측에 사용



LSTM (Long Short Term Memory)

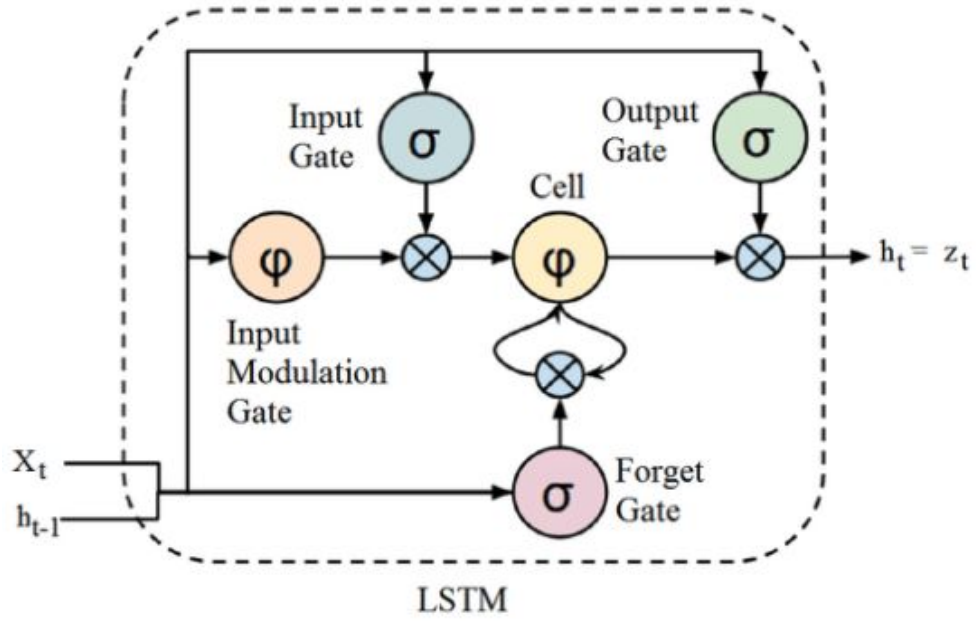
- A type of RNN that introduces the memory cell, a unit of computation that replaces traditional artificial neurons in the hidden layer of the network
  RNN의 한종류로 은닉층안에 있는 뉴런 대신 기억셀을 소개



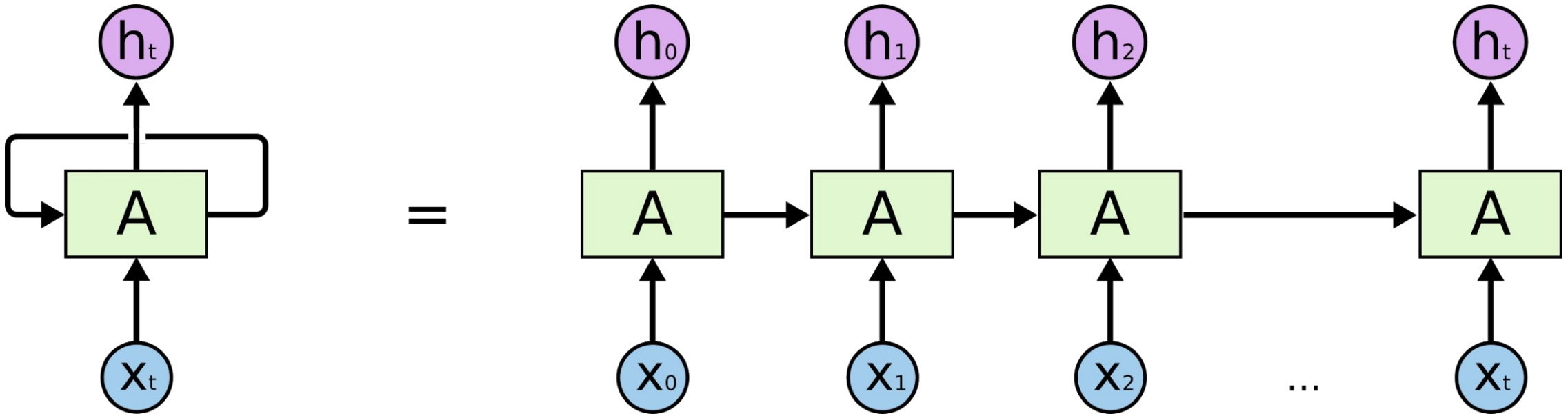
Forget Gate

- Also called remember vector
- The output of the forget gate tells the cell state which information to forget by multiplying 0
  0을 곱해서 기억을 지우거나..
- If the output of the forget gate is 1, the information is kept in the cell state
  1을 곱해서 기억을 남기던가..

LSTM uses a set of gates to control the flow of information
게이트를 이용하여 정보의 흐름을 조절함



Recurrent Cells



Libraries for RNN

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM, CuDNNLSTM
```

Data Preparation for RNN

```
x_train = []
y_train = []
for i in range(60, 1407):
    x_train.append(train[i-60:i, 0])
    y_train.append(train[i, 0])
x_train, y_train = np.array(x_train), np.array(y_train)
```

Reshape Input Values

```
x_train = np.reshape(x_train,
(x_train.shape[0],
x_train.shape[1], 1))

x_test = np.reshape(x_test,
(x_test.shape[0],
x_test.shape[1], 1))
```

- Reshape train x and test x to be [samples, time steps, features]
  훈련과 테스트셋의 모양을 바꿈

MLP Model

```
model = Sequential()
model.add(Dense(100, activation='relu', input_dim=3))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
```

CNN Model

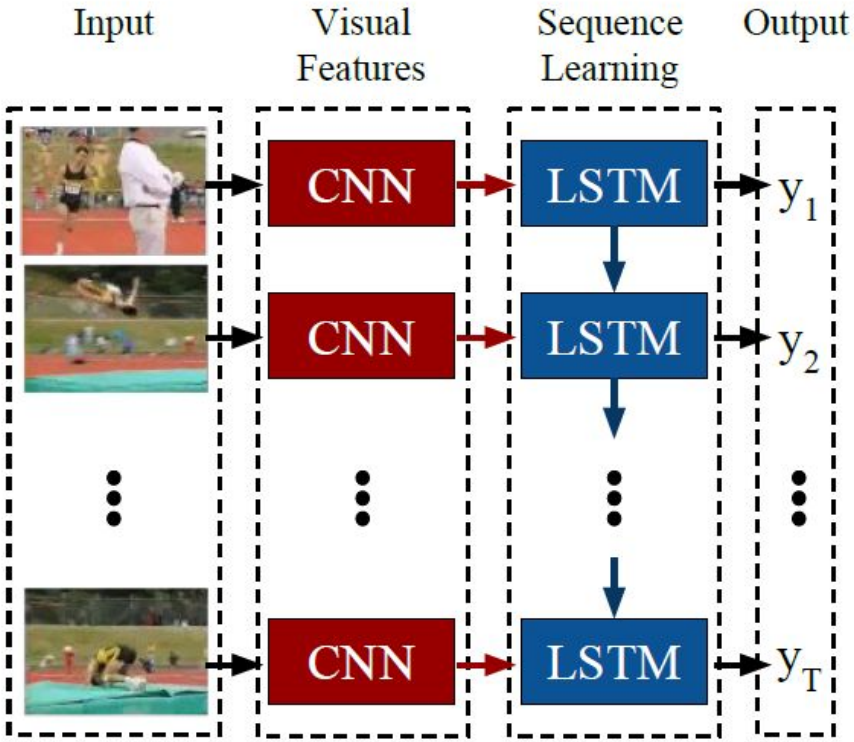
```
model = Sequential()

model.add(Conv1D(filters=64, kernel_size=2,
activation='relu', input_shape=(3, 1)))

model.add(MaxPooling1D(pool_size=2))

model.add(Flatten())

model.add(Dense(50, activation='relu'))

model.add(Dense(1))

model.compile(optimizer='adam', loss='mse')
```

LSTM Model

```
model = Sequential()
model.add(LSTM(50, activation='relu', input_shape=(3, 1)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
```

CNN-LSTM Model

```
model = Sequential()

model.add(TimeDistributed(Conv1
D(filters=64, kernel_size=1,
activation='relu'),
input_shape=(None, 2, 1)))

model.add(TimeDistributed(MaxPo
oling1D(pool_size=2)))

model.add(TimeDistributed(Flatt
en()))

model.add(LSTM(50,
activation='relu'))

model.add(Dense(1))

model.compile(optimizer='adam',
loss='mse')
```



AirPassengers

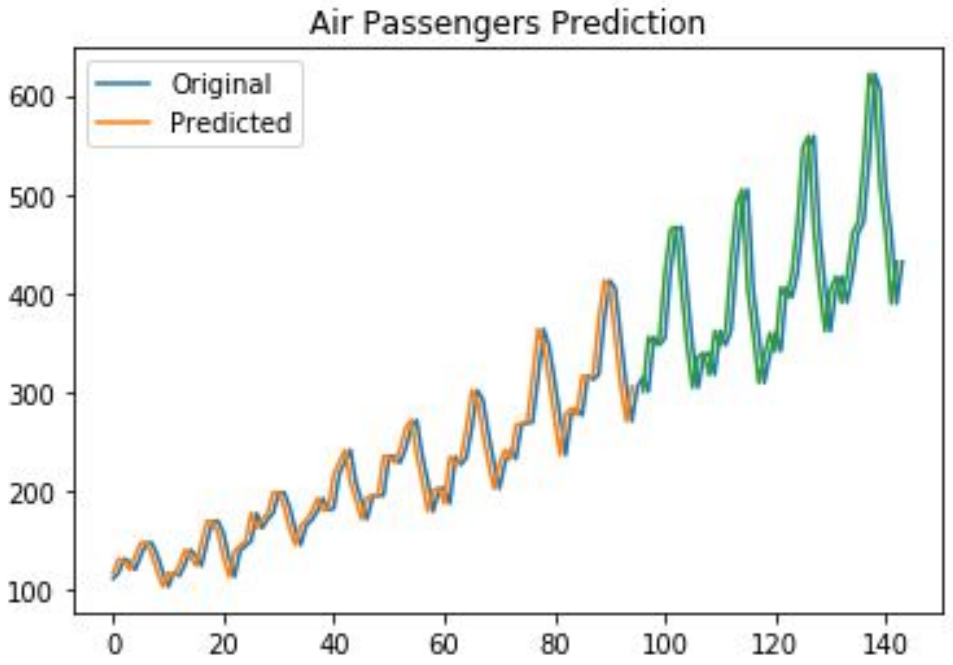
AirPassengers dataset includes monthly Airline Passenger Numbers 1949-1960. Build a prediction model for Airline Passenger Number. **에어패씬저 데이터셋에는 1949년에서 1960년까지의 월별 승객수에 대한 데이터를 포함하고 있습니다. 승객수를 예측하는 모델을 만드시요.**

Exercise #6

- Use AirPassanger Dataset to build a prediction model using RNN
  - Scaling (MinMaxScaler between 0 and 1)
  - Split into train and test datasets
  - Prepare data using shift(1)
  - Reshape input data to be [samples, time steps, features]

Exercise #6

- Build a model
  - 4 units of LSTM
  - 1 Dense layer
  - adam optimizer and mse loss
- Fit the model
  - 100 epochs and 1 batch size
- Prediction using test dataset
- Calculate the root mean squared error
- Visualize the result



Exercise #6

- Collect the Google stock price from yahoo finance.
  - January 2012 to 2018
- Check the null values df.isna().sum()
- Scaling (MinMaxScaler between 0 and 1)
- Preparing train and test sets with 60 timesteps
- Reshaping(x_train.shape[0], x_train.shape[1], 1)

Index	Open	High	Low	Close	Adj Close	Volume
2012-01-03…	326.797	334.409	326.512	333.038	333.038	7345600
2012-01-04…	332.848	335.46	330.641	334.474	334.474	5722200
2012-01-05…	331.396	332.317	328.443	329.835	329.835	6559200
2012-01-06…	329.905	330.33	325.22	325.335	325.335	5380400
2012-01-09…	323.574	323.824	310.926	311.542	311.542	11633500
2012-01-10…	315.19	317.217	308.764	311.882	311.882	8782400
2012-01-11…	312.062	315.01	310.871	313.293	313.293	4795200
2012-01-12…	315.926	316.762	313.564	315.135	315.135	3746600
2012-01-13…	313.443	313.789	310.841	312.808	312.808	4609900

Exercise #6

- Building the RNN LSTM model using sequential model
  - 4 LSTM laysers with 50 units
  - Dropout right after LSTM layers - .2
  - Dense with 1 units
  - Compile with adam optimizer and mean_squared_error loss
  - Fit with 100 epochs and 32 batch size
- Building a RNN model without Dropout

Exercise #6

- Preparing test dataset with 2019 data
  - Concat with original dataset
  - Transform
  - Reshape
  - Predict
  - Inverse_transform
- Visualize the result
- Visualize the rolling mean (30 days)