```
import tensorflow as tf
#from tensorflow.keras.preprocessing.image import ImageDataGenerator -> 안 씀
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization
#from tensorflow.keras.regularizers import l2
from tensorflow.keras.optimizers import Adam
import numpy as np
import matplotlib.pyplot as plt
```

```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()

x_train.shape, x_test.shape

#x_train, y_train = x_train[:10000]
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 4s 0us/step
((50000, 32, 32, 3), (10000, 32, 32, 3))
```

```
x_train, y_train = x_train[:12000], y_train[:12000]
```

```
x_train
```

```
        [ 65,  30,  33]],
```

```
x_train.shape, y_train.shape
```

```
((12000, 32, 32, 3), (12000, 1))
```

제안된 코드에 라이선스가 적용될 수 있습니다.

```
model = Sequential()

model.add(Conv2D(32, 3, activation='relu', padding = 'same', input_shape = x_train.shape[1:]))  # 필터 32개, 필터 크기 3
model.add(Conv2D(32, 3, padding='same', activation='relu'))  # 이미지 크기 똑같이. padding = 'same'
model.add(MaxPooling2D())  # default 2 x 2, 이미지 크기 절반.
model.add(Dropout(0.25))



model.add(Conv2D(128, 3, padding = 'same', activation = 'relu'))
model.add(Conv2D(128, 3, padding = 'same', activation = 'relu'))
# 5줄 추가
model.add(Conv2D(128, 5, padding = 'same', activation = 'relu'))
model.add(Conv2D(256, 3, padding = 'same', activation = 'relu'))
model.add(Conv2D(256, 5, padding = 'same', activation = 'relu'))
model.add(Conv2D(512, 3, padding = 'same', activation = 'relu'))
model.add(Conv2D(512, 5, padding = 'same', activation = 'relu'))

#model.add(GlobalAveragePooling2D())  # FC 없애려고 도입.
model.add(Dropout(0.25))



model.add(BatchNormalization())
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
#model.add(Dense(256, activation = 'relu'))
#model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))

model.summary()
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 32)	896
conv2d_1 (Conv2D)	(None, 32, 32, 32)	9,248
max_pooling2d (MaxPooling2D)	(None, 16, 16, 32)	0
dropout (Dropout)	(None, 16, 16, 32)	0
conv2d_2 (Conv2D)	(None, 16, 16, 128)	36,992
conv2d_3 (Conv2D)	(None, 16, 16, 128)	147,584
conv2d_4 (Conv2D)	(None, 16, 16, 128)	409,728
conv2d_5 (Conv2D)	(None, 16, 16, 256)	295,168
conv2d_6 (Conv2D)	(None, 16, 16, 256)	1,638,656
conv2d_7 (Conv2D)	(None, 16, 16, 512)	1,180,160
conv2d_8 (Conv2D)	(None, 16, 16, 512)	6,554,112
dropout_1 (Dropout)	(None, 16, 16, 512)	0
batch_normalization (BatchNormalization)	(None, 16, 16, 512)	2,048
flatten (Flatten)	(None, 131072)	0
dense (Dense)	(None, 512)	67,109,376
dropout_2 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 10)	5,130

Total params: 77,389,098 (295.22 MB)
Trainable params: 77,388,074 (295.21 MB)
Non-trainable params: 1,024 (4.00 KB)

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

```
history = model.fit(x_train, y_train, epochs = 30, batch_size = 32)
```

```
Epoch 2/30
375/375 ━━━━━━━━━━━━━━━━━━━━ 68s 90ms/step - accuracy: 0.0993 - loss: 2.3026
Epoch 3/30
375/375 ━━━━━━━━━━━━━━━━━━━━ 42s 91ms/step - accuracy: 0.0987 - loss: 2.3056
Epoch 4/30
375/375 ━━━━━━━━━━━━━━━━━━━━ 34s 90ms/step - accuracy: 0.0980 - loss: 2.3035
Epoch 5/30
375/375 ━━━━━━━━━━━━━━━━━━━━ 41s 91ms/step - accuracy: 0.1002 - loss: 2.3027
Epoch 6/30
375/375 ━━━━━━━━━━━━━━━━━━━━ 41s 91ms/step - accuracy: 0.0999 - loss: 2.3025
Epoch 7/30
375/375 ━━━━━━━━━━━━━━━━━━━━ 41s 91ms/step - accuracy: 0.1014 - loss: 2.3024
Epoch 8/30
375/375 ━━━━━━━━━━━━━━━━━━━━ 41s 91ms/step - accuracy: 0.0994 - loss: 2.3037
Epoch 9/30
375/375 ━━━━━━━━━━━━━━━━━━━━ 41s 91ms/step - accuracy: 0.1015 - loss: 2.3055
Epoch 10/30
375/375 ━━━━━━━━━━━━━━━━━━━━ 41s 91ms/step - accuracy: 0.1053 - loss: 2.3162
Epoch 11/30
375/375 ━━━━━━━━━━━━━━━━━━━━ 34s 91ms/step - accuracy: 0.1052 - loss: 2.3026
Epoch 12/30
375/375 ━━━━━━━━━━━━━━━━━━━━ 41s 90ms/step - accuracy: 0.1022 - loss: 2.3023
Epoch 13/30
375/375 ━━━━━━━━━━━━━━━━━━━━ 41s 90ms/step - accuracy: 0.0953 - loss: 2.3029
Epoch 14/30
375/375 ━━━━━━━━━━━━━━━━━━━━ 41s 90ms/step - accuracy: 0.1004 - loss: 2.3022
Epoch 15/30
375/375 ━━━━━━━━━━━━━━━━━━━━ 41s 90ms/step - accuracy: 0.1047 - loss: 2.3023
Epoch 16/30
375/375 ━━━━━━━━━━━━━━━━━━━━ 41s 91ms/step - accuracy: 0.1047 - loss: 2.3031
Epoch 17/30
375/375 ━━━━━━━━━━━━━━━━━━━━ 41s 91ms/step - accuracy: 0.1007 - loss: 2.3029
```

```
Epoch 19/30
375/375 ━━━━━━━━━━━━━━━━━━━━ 41s 91ms/step - accuracy: 0.0983 - loss: 2.3030
Epoch 20/30
375/375 ━━━━━━━━━━━━━━━━━━━━ 41s 91ms/step - accuracy: 0.1029 - loss: 2.3021
Epoch 21/30
375/375 ━━━━━━━━━━━━━━━━━━━━ 41s 90ms/step - accuracy: 0.1077 - loss: 2.3025
Epoch 22/30
375/375 ━━━━━━━━━━━━━━━━━━━━ 41s 91ms/step - accuracy: 0.0995 - loss: 2.3025
Epoch 23/30
375/375 ━━━━━━━━━━━━━━━━━━━━ 34s 90ms/step - accuracy: 0.1058 - loss: 2.3026
Epoch 24/30
375/375 ━━━━━━━━━━━━━━━━━━━━ 41s 91ms/step - accuracy: 0.1020 - loss: 2.3027
Epoch 25/30
375/375 ━━━━━━━━━━━━━━━━━━━━ 41s 90ms/step - accuracy: 0.0987 - loss: 2.3026
Epoch 26/30
375/375 ━━━━━━━━━━━━━━━━━━━━ 41s 90ms/step - accuracy: 0.0994 - loss: 2.3024
Epoch 27/30
375/375 ━━━━━━━━━━━━━━━━━━━━ 41s 90ms/step - accuracy: 0.1002 - loss: 2.3023
Epoch 28/30
375/375 ━━━━━━━━━━━━━━━━━━━━ 41s 90ms/step - accuracy: 0.1007 - loss: 2.3027
Epoch 29/30
375/375 ━━━━━━━━━━━━━━━━━━━━ 41s 90ms/step - accuracy: 0.1048 - loss: 2.3020
Epoch 30/30
375/375 ━━━━━━━━━━━━━━━━━━━━ 41s 90ms/step - accuracy: 0.0999 - loss: 2.3026
```

```
plt.subplot(121)
plt.plot(history.history['loss'], label='train loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Model train Loss')

plt.subplot(122)
plt.plot(history.history['accuracy'], label='train accuracy')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Model train accuracy')

plt.legend()
plt.show()
```