CMT: Convolutional Neural Networks Meet Vision Transformer
- 실습

이찬호
국방인공지능융합연구소

CONTENTS

1. DATASETS - CIFAR-10

❑ CIFAR-10 (Canadian Institute for Advanced Research-10)

- 해상도 : 32 X 32 X 3
- 클래스 : 10
- 학습 데이터 수 : 50,000장 (클래스 당 5,000장)
- 테스트 데이터 수 : 10,000장 (클래스 당 1,000장)

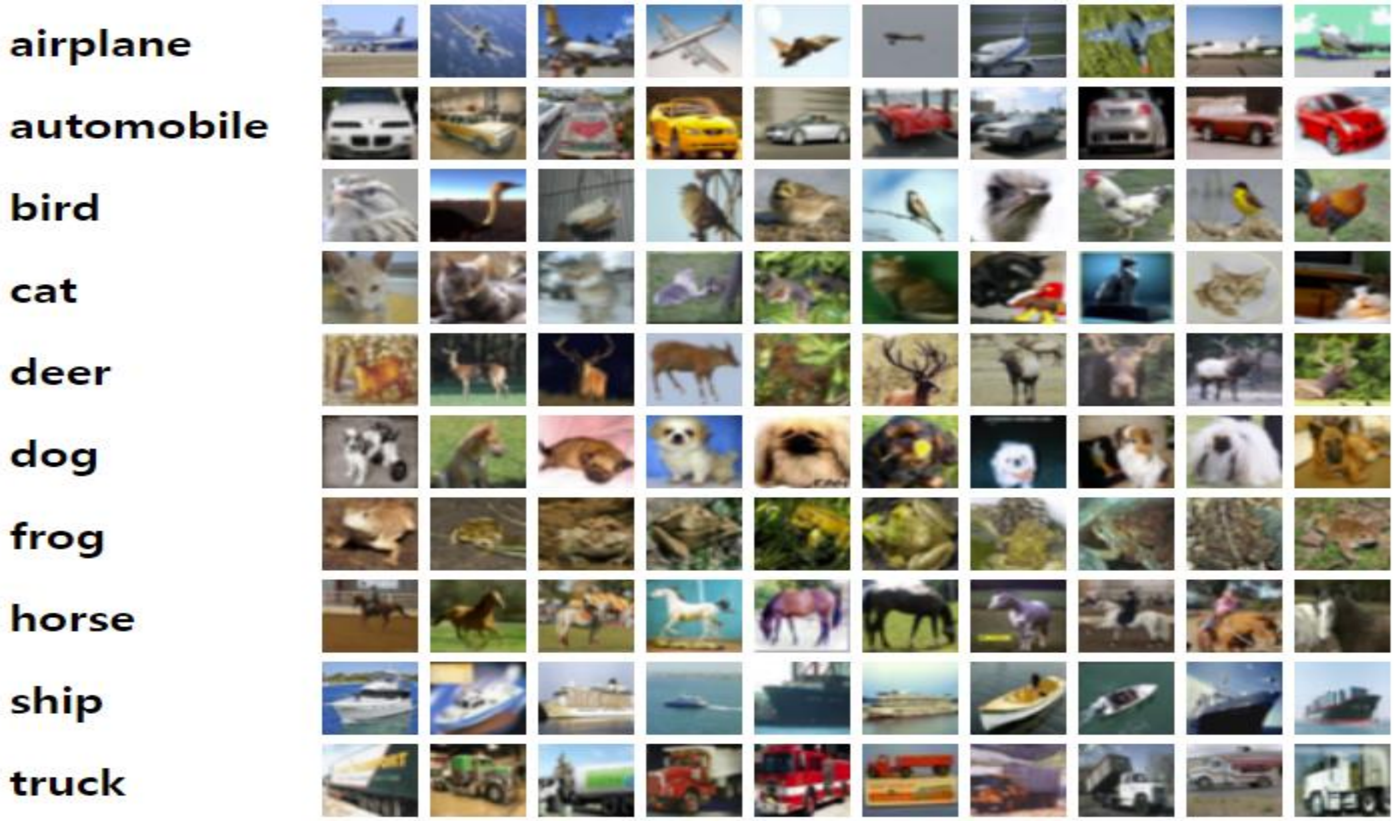


FIG 1. CIFAR-10 Datasets 일부

1. DATASETS - ImageNet-100

❑ ImageNet-100

- 클래스 : 100
- 학습 데이터 수 : 130,000장 (클래스 당 1,300장)
- 테스트 데이터 수 : 5,000장 (클래스 당 50장)

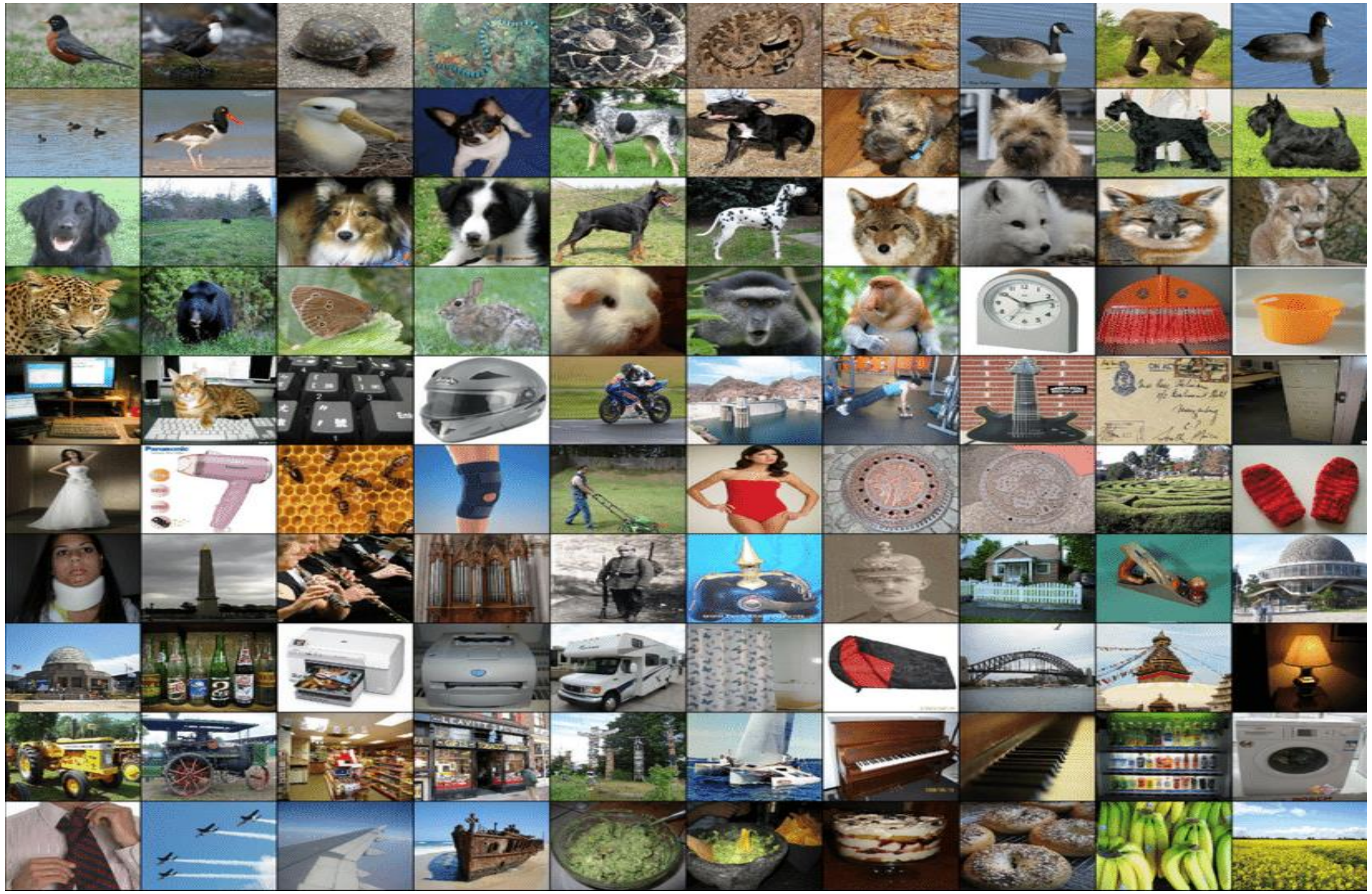


FIG 2. ImageNet-100 Datasets 일부

2. CMT ARCHITECTURE & CODE

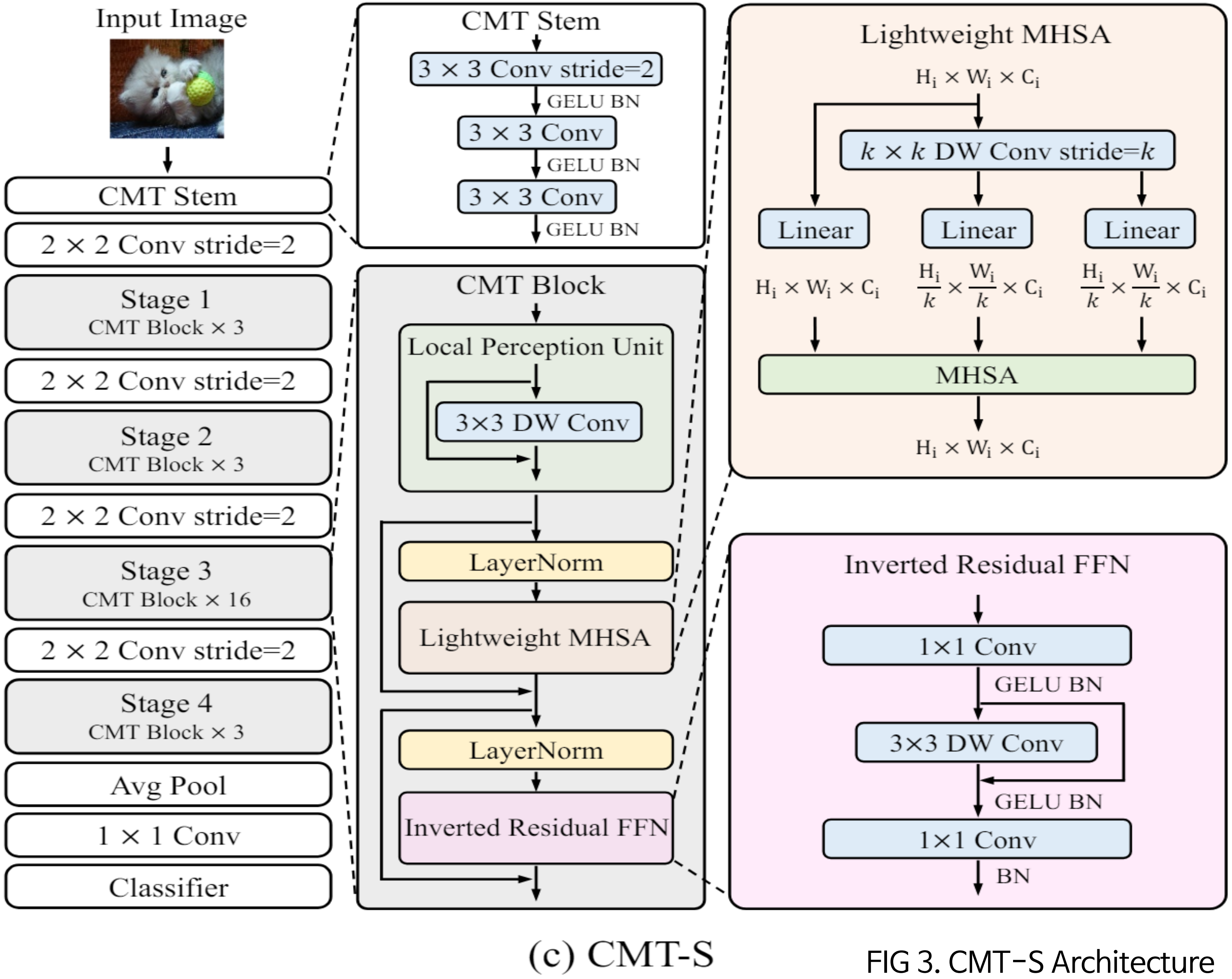
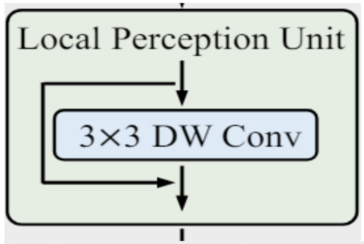


FIG 3. CMT-S Architecture

2. CMT ARCHITECTURE & CODE - LPU

❑ Local Perception Unit (LPU)

- Depth-Wise Conv를 수행한 결과를 입력 데이터에 더해, 입력의 원래 정보를 보존하면서 추가 특징 학습




```python
class LocalPerceptionUnit(Layer):
    def __init__(self, filters):
        super(LocalPerceptionUnit, self).__init__()
        self.dw_conv = DepthwiseConv2D(kernel_size=3, padding='same')
        self.filters = filters

    def call(self, x):
        return x + self.dw_conv(x)

    def get_config(self):
        config = super().get_config()
        config.update({"filters": self.filters})
        return config
```

FIG 4. LPU Code

2. CMT ARCHITECTURE & CODE - LMHSA

❑ Lightweight Multi Head Self-Attention (LMHSA)

- 입력 데이터 텐서에 대해 경량화 된 멀티 헤드 셀프 어텐션을 수행하여 각 공간 위치에서 중요한 특징을 강조하여 반환

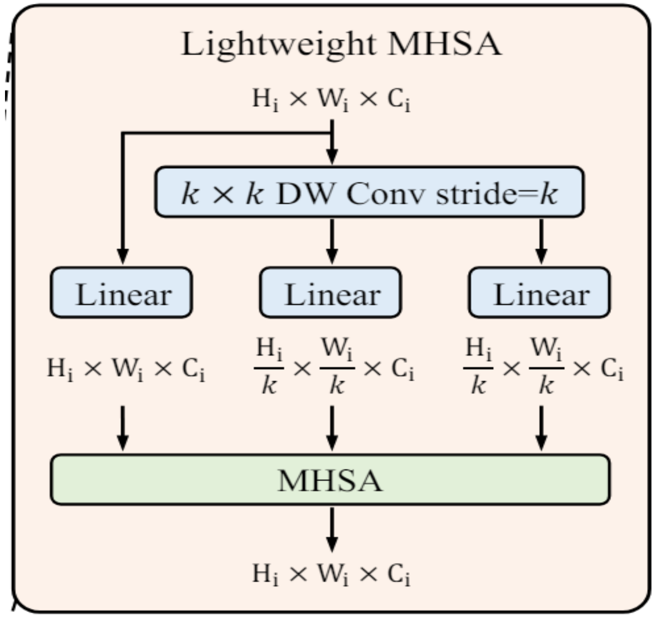


FIG 5. LMHSA Code

```python
class LightweightMHSA(Layer):
    def __init__(self, dim, num_heads=8):
        super(LightweightMHSA, self).__init__()
        self.num_heads = num_heads
        self.dim = dim

    def build(self, input_shape):
        self.qkv = Dense(self.dim * 3)
        self.proj = Dense(self.dim)

    def call(self, x):
        B, H, W, C = tf.shape(x)[0], tf.shape(x)[1], tf.shape(x)[2], self.dim
        N = H * W  # Flattened spatial dimensions
        qkv = self.qkv(x)
        qkv = tf.reshape(qkv, (B, N, 3, self.num_heads, C // self.num_heads))
        qkv = tf.transpose(qkv, perm=[2, 0, 3, 1, 4])
        q, k, v = qkv[0], qkv[1], qkv[2]

        attn = tf.nn.softmax(tf.matmul(q, k, transpose_b=True) / (C // self.num_heads) ** 0.5, axis=-1)
        attn = tf.matmul(attn, v)

        attn = tf.transpose(attn, perm=[0, 2, 1, 3])
        attn = tf.reshape(attn, (B, H, W, C))

        return self.proj(attn)

    def get_config(self):
        config = super().get_config()
        config.update({"dim": self.dim, "num_heads": self.num_heads})
        return config
```

2. CMT ARCHITECTURE & CODE - IRFFN

❑ Inverted Residual Feed-Forward Network (IRFFN)

- 입력 데이터를 확장 후, Depth-Wise Conv와 BN을 통해 처리하고, 원래 입력 데이터에 다시 추가
- 이 과정은 모델의 표현력을 높이고, 더 복잡한 특징 학습

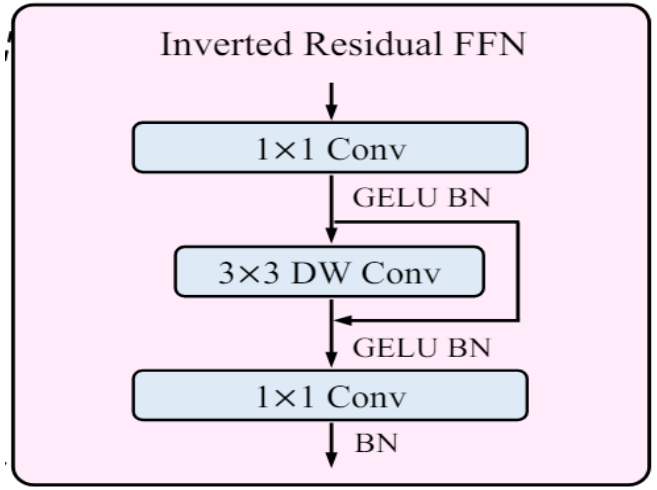


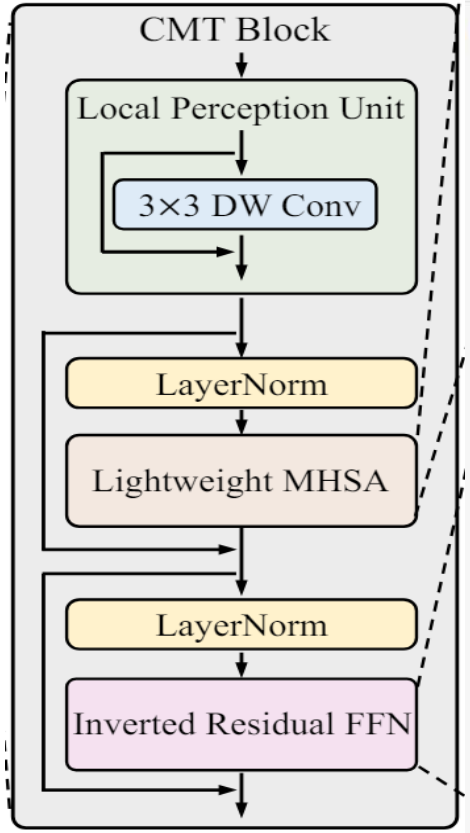
FIG 6. IRFFN Code

```
class InvertedResidualFFN(Layer):
    def __init__(self, dim, expansion_ratio=4):
        super(InvertedResidualFFN, self).__init__()
        self.hidden_dim = int(dim * expansion_ratio)
        self.expand = Dense(self.hidden_dim)
        self.depthwise = DepthwiseConv2D(kernel_size=3, padding='same')
        self.project = Dense(dim)
        self.bn1 = BatchNormalization()
        self.bn2 = BatchNormalization()
        self.gelu = Activation('gelu')

    def call(self, x):
        residual = x
        x = self.expand(x)
        x = self.bn1(x)
        x = self.gelu(x)
        x = self.depthwise(x)
        x = self.bn2(x)
        x = self.project(x)
        return residual + x

    def get_config(self):
        config = super().get_config()
        config.update({"dim": self.hidden_dim // 4, "expansion_ratio": 4})
        return config
```

2. CMT ARCHITECTURE & CODE - CMT Block

❑ CMT Block

- 입력 데이터에 LPU, LMHSA, IRFFN을 차례로 적용하고, 각 연산 후 입력 데이터에 더해주는 구조
- 이를 통해 입력 데이터의 다양한 특징을 효과적으로 학습



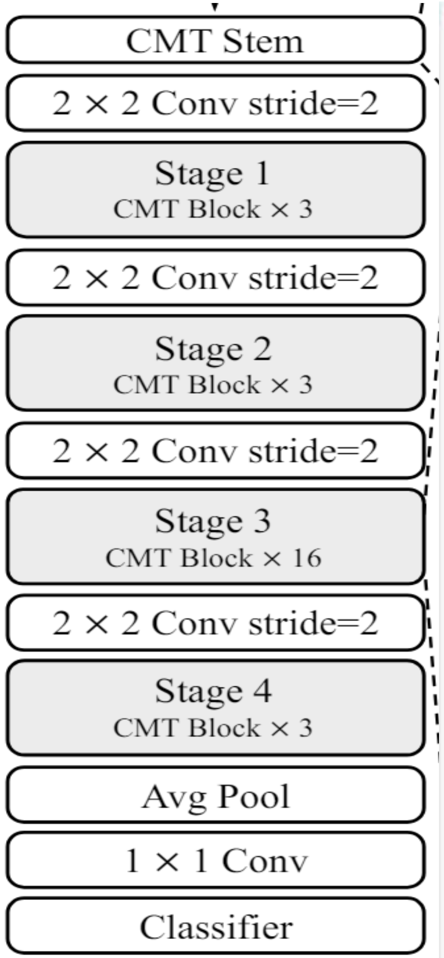

```python
class CMTBlock(Layer):
    def __init__(self, dim, num_heads=8, mlp_ratio=4.):
        super(CMTBlock, self).__init__()
        self.lpu = LocalPerceptionUnit(dim)
        self.lmhsa = LightweightMHSA(dim, num_heads)
        self.irffn = InvertedResidualFFN(dim, mlp_ratio)
        self.norm1 = LayerNormalization()
        self.norm2 = LayerNormalization()

    def call(self, x):
        x = self.lpu(x)
        x = x + self.lmhsa(self.norm1(x))
        x = x + self.irffn(self.norm2(x))
        return x

    def get_config(self):
        config = super().get_config()
        config.update({
            "dim": self.lpu.filters,
            "num_heads": self.lmhsa.num_heads,
            "mlp_ratio": self.irffn.hidden_dim // self.lpu.filters
        })
        return config
```

FIG 7. CMT Block Code

2. CMT ARCHITECTURE & CODE - CMT-S Model

❑ CMT-S Model

- 각 단계에서는 입력 데이터를 점진적으로 다운 샘플링하고 CMT Block으로 입력 데이터의 다양한 특징을 학습
- 마지막 단계에선 Global Average Pooling과 Dense를 통해 Class 예측




```python
def cmt_s(input_shape=(224, 224, 3), num_classes=1000):
    inputs = Input(shape=input_shape)

    # Stem
    x = Conv2D(32, kernel_size=3, strides=2, padding='same')(inputs)
    x = BatchNormalization()(x)
    x = Activation('gelu')(x)
    x = Conv2D(32, kernel_size=3, padding='same')(x)
    x = BatchNormalization()(x)
    x = Activation('gelu')(x)
    x = Conv2D(32, kernel_size=3, padding='same')(x)
    x = BatchNormalization()(x)
    x = Activation('gelu')(x)

    # Stage 1
    x = Conv2D(64, kernel_size=2, strides=2, padding='same')(x)
    x = BatchNormalization()(x)
    x = Activation('gelu')(x)
    for _ in range(3):
        x = CMTBlock(64)(x)

    # Stage 2
    x = Conv2D(128, kernel_size=2, strides=2, padding='same')(x)
    x = BatchNormalization()(x)
    x = Activation('gelu')(x)
    for _ in range(3):
        x = CMTBlock(128)(x)

    # Stage 3
    x = Conv2D(256, kernel_size=2, strides=2, padding='same')(x)
    x = BatchNormalization()(x)
    x = Activation('gelu')(x)
    for _ in range(16):
        x = CMTBlock(256)(x)

    # Stage 4
    x = Conv2D(512, kernel_size=2, strides=2, padding='same')(x)
    x = BatchNormalization()(x)
    x = Activation('gelu')(x)
    for _ in range(3):
        x = CMTBlock(512)(x)

    x = GlobalAveragePooling2D()(x)

    x = Dense(1280)(x)
    x = BatchNormalization()(x)
    x = Activation('gelu')(x)

    outputs = Dense(num_classes, activation='softmax')(x)

    model = Model(inputs, outputs)
    return model
```

FIG 8. CMT-S Code

3. Result - CIFAR-10

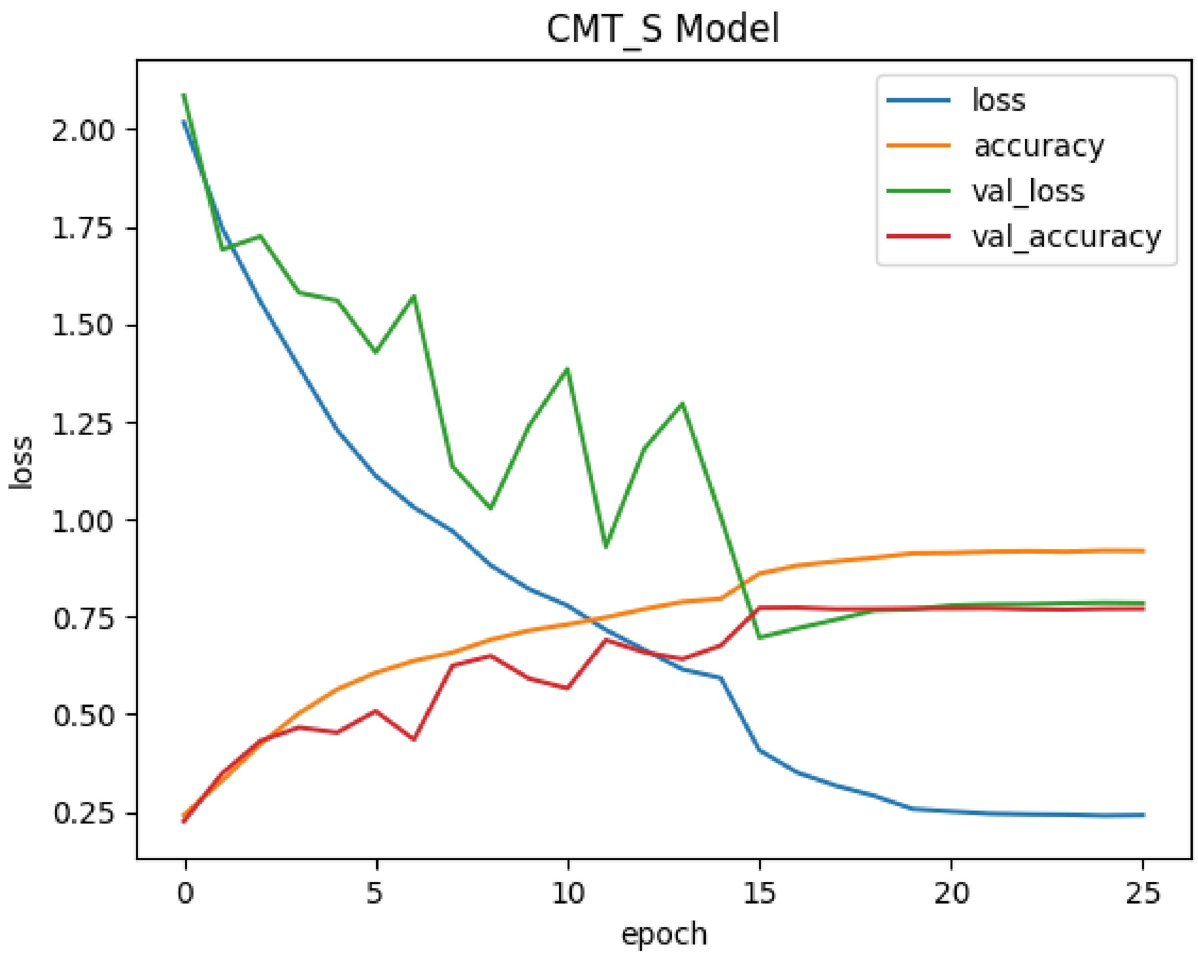


FIG 9. CMT-S Train Result by CIFAR-10

Datasets	CIFAR-10
Optimizer	Adam
Learning Rate	1e-3(0.001)
Weight Decay	-
Epoch	26
Loss	0.4076
Accuracy	0.8601
Validation Loss	0.6962
Validation Accuracy	0.7720
Test Loss	0.8227
Test Accuracy	0.7641

3. Result - IamgeNet-100

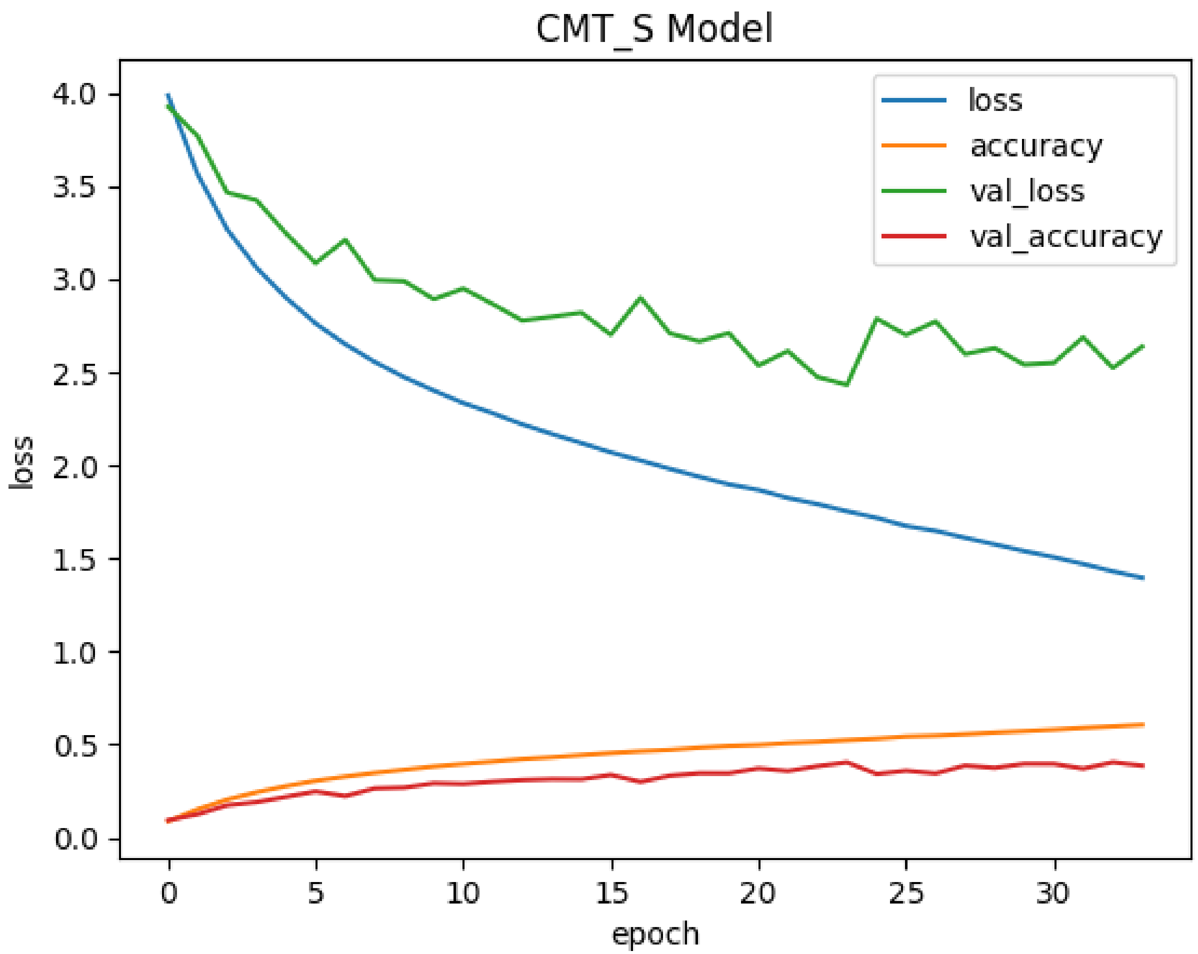


FIG 10. CMT-S Train Result by ImageNet-100

Datasets	ImageNet-100
Optimizer	Adam
Learning Rate	1e-3(0.001)
Weight Decay	5e-9(0.00005)
Epoch	34
Loss	1.7521
Accuracy	0.5230
Validation Loss	2.4309
Validation Accuracy	0.4042

4. Compare ViT & CMT - CIFAR-10

Datasets	CIFAR-10
Model	ViT
Optimizer	Adam
Learning Rate	1e-3(0.001)
Weight Decay	5e-9(0.000000009)
Epoch	36
Loss	0.8460
Accuracy	0.6919
Validation Loss	1.0870
Validation Accuracy	0.6422
Test Loss	1.1141
Test Accuracy	0.6273

Datasets	CIFAR-10
Model	CMT-S
Optimizer	Adam
Learning Rate	1e-3(0.001)
Weight Decay	-
Epoch	26
Loss	0.4076
Accuracy	0.8601
Validation Loss	0.6962
Validation Accuracy	0.7720
Test Loss	0.8227
Test Accuracy	0.7641

4. Compare ViT & CMT - ImageNet-100

Datasets	ImageNet-100
Model	ViT
Optimizer	Adam
Learning Rate	1e-3(0.001)
Weight Decay	5e-9(0.00005)
Epoch	45
Loss	1.8012
Accuracy	0.5104
Validation Loss	2.4765
Validation Accuracy	0.4110

Datasets	ImageNet-100
Model	CMT-S
Optimizer	Adam
Learning Rate	1e-3(0.001)
Weight Decay	5e-9(0.00005)
Epoch	34
Loss	1.7521
Accuracy	0.5230
Validation Loss	2.4309
Validation Accuracy	0.4042

THANKS FOR YOUR ATTENTION

이찬호
국방인공지능융합연구소