Improving CNNs and Model Interpretability

Chapter advanced techniques untuk production CNNs: regularization, Minception architecture, transfer learning, dan Grad-CAM interpretability. Fokus pada mengurangi overfitting dan model explanation. [1]

Comprehensive Overfitting Reduction Techniques

1. Data Augmentation Pipeline:

```
def advanced_augmentation(image, label):
    # Geometric transforms
    image = tf.image.random_flip_left_right(image)
    image = tf.image.random_brightness(image, 0.2)
    image = tf.image.random_contrast(image, 0.8, 1.2)

    # Advanced: Cutout/Random Erasing
    if tf.random.uniform(()) > 0.5:
        size = tf.random.uniform((), 0, 20, dtype=tf.int32)
        x = tf.random.uniform((), 0, 224-size, dtype=tf.int32)
        y = tf.random.uniform((), 0, 224-size, dtype=tf.int32)
        image = tf.tensor_scatter_nd_update(
            image, [[x, y, 0], [x+size, y, 2]],
            tf.zeros([size, size, 3], image.dtype)
        )
    return image, label
```

2. Dropout dan Regularization:

```
model = Sequential([
    Conv2D(64, 3, activation='relu'),
    BatchNormalization(),
    Dropout(0.3),  # Spatial dropout untuk conv layers
    Conv2D(128, 3, activation='relu'),
    MaxPool2D(),
    # ...
])
```

3. Early Stopping + Model Checkpointing:

```
callbacks = [
    tf.keras.callbacks.EarlyStopping(
        monitor='val_loss', patience=10, restore_best_weights=True
    ),
    tf.keras.callbacks.ModelCheckpoint('best_model.h5', save_best_only=True)
]
```

Minception Architecture Inception-ResNet Hybrid)

Stem: Multi-branch conv + residual connection
Inception-ResNet-A/B Blocks: Inception modules + identity mapping **Reduction Blocks**: Dimensionality reduction dengan atrous conv

```
def inception_resnet_a(inputs):
    # Inception branches
    branch_1 = Conv2D(32, 1, activation='relu')(inputs)
    branch_3a = Conv2D(32, 1)(inputs); branch_3a = Conv2D(32, 3)(branch_3a)
    branch_3b = Conv2D(32, 1)(inputs); branch_3b = Conv2D(32, 3)(branch_3b); branch_3b =

    mixed = Concatenate()([branch_1, branch_3a, branch_3b])
    up = Conv2D(256, 1, activation='relu')(mixed)

    # Residual connection
    return tf.keras.layers.add([inputs, up])
```

Transfer Learning: Feature Extraction + Fine-tuning

```
# Load pretrained ResNet50
base_model = tf.keras.applications.ResNet50(
    weights='imagenet', include_top=False, pooling='avg'
)
base_model.trainable = False  # Freeze weights

model = Sequential([
    base_model,
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(num_classes, activation='softmax')
])

# Fine-tuning tahap 2: Unfreeze top layers
base_model.trainable = True
for layer in base_model.layers[:-30]:
    layer.trainable = False
```

Grad-CAM Model Interpretability

Gradient-weighted Class Activation Mapping:

```
def make_gradcam_heatmap(img_array, model, last_conv_layer_name, pred_index=None):
    grad_model = tf.keras.models.Model(
        [model.inputs],
        [model.get_layer(last_conv_layer_name).output, model.output]
    )

    with tf.GradientTape() as tape:
        last_conv_layer_output = grad_model(img_array)
        preds = last_conv_layer_output[^1]
        class_channel = preds[:, pred_index]
    grads = tape.gradient(class_channel, last_conv_layer_output[^0])
    pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))

    conv_layer_out = last_conv_layer_output[^0][^0]
    heatmap = conv_layer_out @ pooled_grads[..., tf.newaxis]
    heatmap = tf.squeeze(heatmap)
    heatmap = tf.maximum(heatmap, 0) / tf.math.reduce_max(heatmap)
    return heatmap
```

Kesimpulan

Production CNNs membutuhkan regularization komprehensif, transfer learning, dan interpretability tools untuk deployment yang bertanggung jawab.[1]

```
def get_segmentation_pipeline(image_paths, mask_paths, batch_size=8):
    dataset = tf.data.Dataset.from_tensor_slices((image_paths, mask_paths))
    dataset = dataset.map(preprocess_image, num_parallel_calls=AUTOTUNE)

    # Cache + shuffle + repeat + batch + prefetch
    dataset = dataset.cache()
    dataset = dataset.shuffle(1000)
```