Parallel Visual Attention Encoder for Model Improvement

Dog Breeds Predictor

Fernando Martínez

Inspiration of the project

New York City is one of the most diverse cities in the world when it comes to dog breeds we see everyday.

Back in Summer, walking in Central Park, I was surprised that I couldn't recognize many of the dog breeds I was seeing.

Approach to address the problem

Developed a model based on the **EfficientNet** family of Deep Learning architectures to train an effective but efficient model. The proposed architecture improves its prediction power using an attention branch in parallel to the base model. This brach simulates an encoder, reducing dimensionality, and then applying an **Attention Module**.

These Attention Module behave as the "latent layer" and combines two innovative attention techniques, "Convolutional Block Attention Module" (CBAM) and "Large Kernel Attention" (LKA) the structure holds efficiency while improving accuracy.

I named this architecture **FerNet** (Fernando's Network).

(Disclaimer: Fernet is an Italian type of amaro, a bitter, aromatic spirit.)

Dataset

- **Stanford Dogs Dataset**
  - Number of categories/classes: **120**
  - Number of Images: **20,580 images**
  - Training split setting**: 80% training (**16,464 images**), 20% test (**4,116 images**).**
- **Tsinghua Dogs Dataset**
  - Number of categories/classes: **130**
  - Number of Images: **70,432 images**
  - Training split setting**: 80% training (**56,346 images**), 20% test (**14,086 images**).**

Architecture Summary (I)

This version of **FerNet** uses **EfficientNetB2** as its main predictor driver. For this exercise I applied transfer learning and fine-tuning towards maintaining the model's feature extraction trained capabilities.

Model: "FerNet"

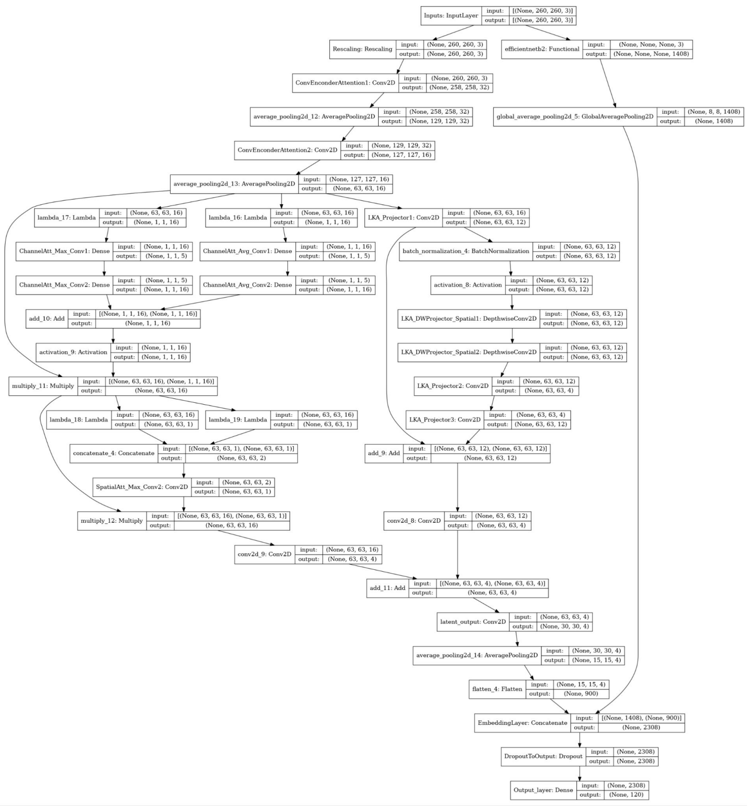
Layer (type)	Output Shape	Param #	Connected to
Inputs (InputLayer)	[(None, 260, 260, 3)	0	
Rescaling (Rescaling)	(None, 260, 260, 3)	0	Inputs[0][0]
ConvEnconderAttention1 (Conv2D)	(None, 258, 258, 32)	896	Rescaling[0][0]
average_pooling2d_12 (AveragePo	(None, 129, 129, 32)	0	ConvEnconderAttention1[0][0]
ConvEnconderAttention2 (Conv2D)	(None, 127, 127, 16)	4624	average_pooling2d_12[0][0]
average_pooling2d_13 (AveragePo	(None, 63, 63, 16)	0	ConvEnconderAttention2[0][0]
lambda_16 (Lambda)	(None, 1, 1, 16)	0	average_pooling2d_13[0][0]
lambda_17 (Lambda)	(None, 1, 1, 16)	0	average_pooling2d_13[0][0]
ChannelAtt_Avg_Conv1 (Dense)	(None, 1, 1, 5)	85	lambda_16[0][0]
ChannelAtt_Max_Conv1 (Dense)	(None, 1, 1, 5)	85	lambda_17[0][0]
LKA_Projector1 (Conv2D)	(None, 63, 63, 12)	204	average_pooling2d_13[0][0]
ChannelAtt_Avg_Conv2 (Dense)	(None, 1, 1, 16)	96	ChannelAtt_Avg_Conv1[0][0]
ChannelAtt_Max_Conv2 (Dense)	(None, 1, 1, 16)	96	ChannelAtt_Max_Conv1[0][0]
batch_normalization_4 (BatchNor	(None, 63, 63, 12)	48	LKA_Projector1[0][0]
add_10 (Add)	(None, 1, 1, 16)	0	ChannelAtt_Avg_Conv2[0][0] ChannelAtt_Max_Conv2[0][0]
activation_8 (Activation)	(None, 63, 63, 12)	0	batch_normalization_4[0][0]
activation_9 (Activation)	(None, 1, 1, 16)	0	add_10[0][0]
LKA_DWProjector_Spatial1 (Depth	(None, 63, 63, 12)	312	activation_8[0][0]
multiply_11 (Multiply)	(None, 63, 63, 16)	0	average_pooling2d_13[0][0] activation_9[0][0]
LKA_DWProjector_Spatial2 (Depth	(None, 63, 63, 12)	600	LKA_DWProjector_Spatial1[0][0]
lambda_18 (Lambda)	(None, 63, 63, 1)	0	multiply_11[0][0]
lambda_19 (Lambda)	(None, 63, 63, 1)	0	multiply_11[0][0]
LKA_Projector2 (Conv2D)	(None, 63, 63, 4)	52	LKA_DWProjector_Spatial2[0][0]
concatenate_4 (Concatenate)	(None, 63, 63, 2)	0	lambda_18[0][0] lambda_19[0][0]
LKA_Projector3 (Conv2D)	(None, 63, 63, 12)	60	LKA_Projector2[0][0]
SpatialAtt_Max_Conv2 (Conv2D)	(None, 63, 63, 1)	98	concatenate_4[0][0]
add_9 (Add)	(None, 63, 63, 12)	0	LKA_Projector3[0][0] LKA_Projector1[0][0]
multiply_12 (Multiply)	(None, 63, 63, 16)	0	multiply_11[0][0] SpatialAtt_Max_Conv2[0][0]
conv2d_8 (Conv2D)	(None, 63, 63, 4)	52	add_9[0][0]
conv2d_9 (Conv2D)	(None, 63, 63, 4)	68	multiply_12[0][0]
add_11 (Add)	(None, 63, 63, 4)	0	conv2d_8[0][0] conv2d_9[0][0]
latent_output (Conv2D)	(None, 30, 30, 4)	404	add_11[0][0]
efficientnetb2 (Functional)	(None, None, None, 1	7768569	Inputs[0][0]
average_pooling2d_14 (AveragePo	(None, 15, 15, 4)	0	latent_output[0][0]
global_average_pooling2d_5 (Glo	(None, 1408)	0	efficientnetb2[0][0]
flatten_4 (Flatten)	(None, 900)	0	average_pooling2d_14[0][0]
EmbeddingLayer (Concatenate)	(None, 2308)	0	global_average_pooling2d_5[0][0] flatten_4[0][0]
DropoutToOutput (Dropout)	(None, 2308)	0	EmbeddingLayer[0][0]
Output_layer (Dense)	(None, 120)	277080	DropoutToOutput[0][0]

Total params: 8,053,429
Trainable params: 284,836
Non-trainable params: 7,768,593

Model Summary - Stanford Dogs Dataset

Architecture Summary (II)

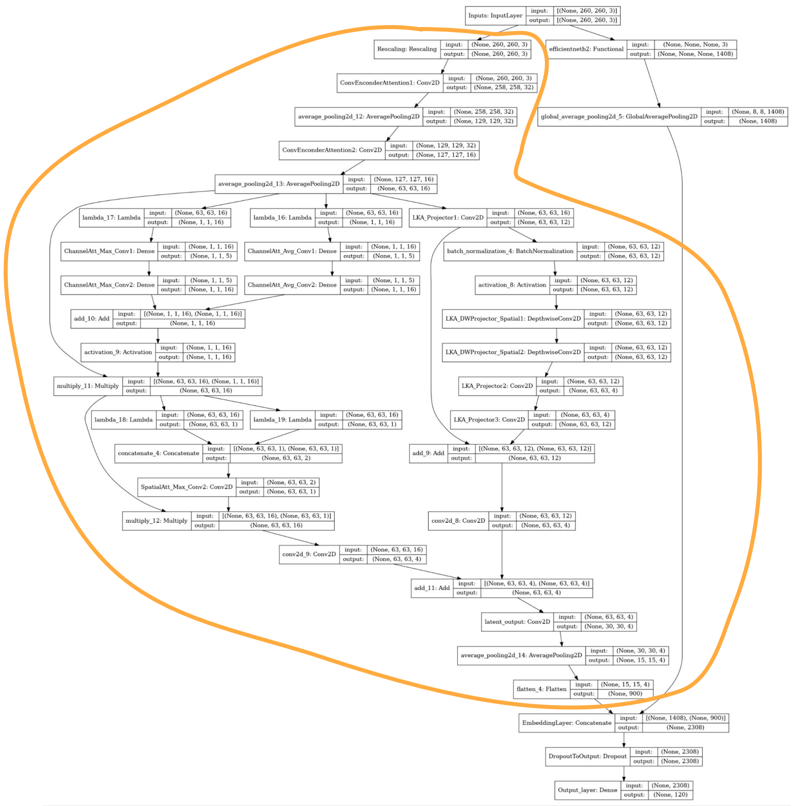
FerNet uses two branches, one is the state-of-the-art model itself (base model), but the magic comes after taking the same input and passing it through another set of layers focused on visual attention. This branch passes the input in parallel through previously mentioned attention modules, LKA and CBAM.



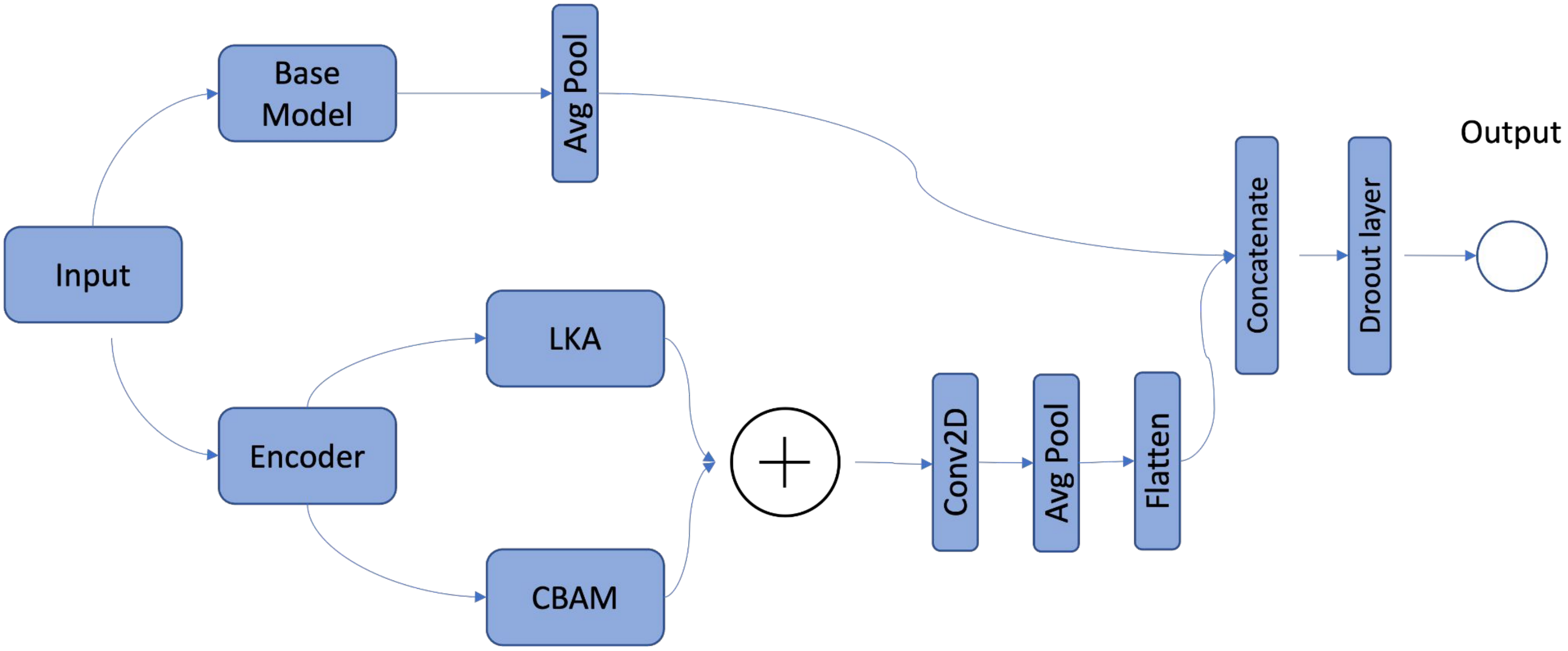
Architecture Summary (II)

FerNet uses two branches, one is the state-of-the-art model itself (base model), but the magic comes after taking the same input and passing it through another set of layers focused on visual attention. This branch passes the input in parallel through previously mentioned attention modules, LKA and CBAM.

After this processing, FerNet adds the attention outputs, and concatenates them with the base model result for further processing.

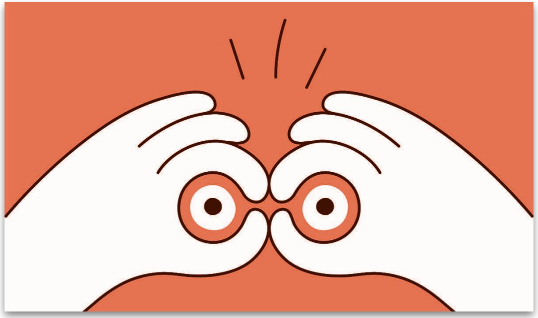


Architecture Summary (III)



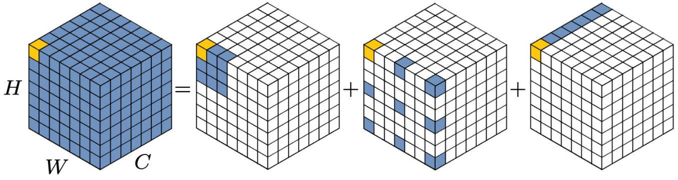
Diving deeper into the attention mechanisms (I)

As humans, **visual attention** is a key aspect of human perception, allowing us to focus on relevant information in our environment while ignoring irrelevant distractions. Similar to what we do with human vision, these machine learning techniques on this topic aim for our models to i**gnore the noise around** objects and concentrate their **attention on what is relevant** to the visual task.

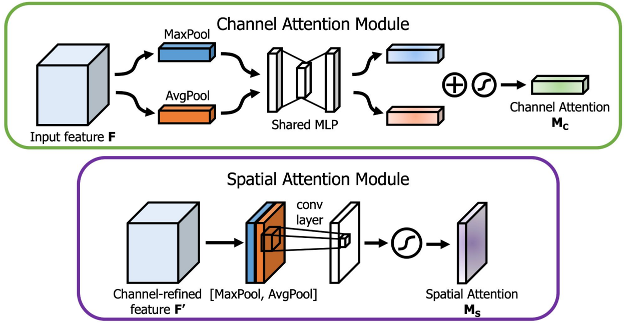


Diving deeper into the attention mechanisms (II)

One of the implemented submodules is **LKA**. This submodule combines the use of depthwise convolutions for spatial attention and regular convolutional layers with a especial setting for channel attention. On the other hand, **CBAM** treat each attention task as separated submodules and execute them in sequence (First channel attention, then spatial attention). The key takeaway of attention modules is that **they create attention maps** indicating **the importance of different parts.**



LKA Representation



CBAM Submodules Representation

Techniques used running our models

- **Distributed Parallel Training:** Train our models using the two (2) GPU NVIDIA T4 provided by the Kaggle Environment.
- **Feature extraction and Fine-tuning:** Executed a sequence of iterations with our base model layers without training and only letting our new branch train. Then, run for the same number of epochs with the last 10% of the base model layers updating their weights.
- **Adam Optimizer** (default settings)
- **Cross-entropy** loss function with **label smoothing** (overcome excessive label confidence)
- **Callbacks**:
  - ModelCheckPoint
  - ReduceLROnPlateau
  - EarlyStopping

Sample of the code

Parallel Visual Attention Encoder for Model Improvement - FerNet

```python
# fix random seed for reproducibility
tf.random.set_seed(0)
with strategy.scope():

    base_model = tf.keras.applications.EfficientNetB2(include_top=False)
    base_model.trainable = False
    input_shape = IMG_SIZE+(3,)

    inputs = tf.keras.layers.Input(shape=input_shape, name="Inputs")

    ## EfficientNet
    effNet_model = base_model(inputs, training=False)

    effNet_model = tf.keras.layers.GlobalAveragePooling2D()(effNet_model)
    eff_model = tf.keras.Model(inputs=inputs, outputs=effNet_model, name="EfficientNetB2")

    ## Visual Attention
    va = tf.keras.layers.Rescaling(1/255., name="Rescaling")(inputs)

    va = tf.keras.layers.Conv2D(32, 3, activation=tf.keras.activations.gelu, strides=1, name="ConvEnconderAttention1")(va)
    va = tf.keras.layers.AveragePooling2D()(va)
    va = tf.keras.layers.Conv2D(16, 3, activation=tf.keras.activations.gelu, strides=1, name="ConvEnconderAttention2")(va)
    va = tf.keras.layers.AveragePooling2D()(va)

    # LKA
    va_input = tf.keras.layers.Conv2D(12, 1, name='LKA_Projector1')(va)
    va1 = tf.keras.layers.BatchNormalization()(va_input)
    va1 = tf.keras.layers.Activation(tf.keras.activations.gelu)(va1)
    va1 = tf.keras.layers.DepthwiseConv2D(5, padding='same',name='LKA_DWProjector_Spatial1')(va1)
    va1 = tf.keras.layers.DepthwiseConv2D(7, padding='same', dilation_rate=3, name='LKA_DWProjector_Spatial2')(va1)
    va1 = tf.keras.layers.Conv2D(4, 1, name='LKA_Projector2')(va1)
    va1 = tf.keras.layers.Conv2D(12, 1, name='LKA_Projector3')(va1)
    va1 = tf.keras.layers.Add()([va1, va_input])
    # CBAM
    ## Channel Attention
    input_channels = va.get_shape()[-1]
    avg_pool_channel = tf.keras.layers.Lambda(lambda x: tf.keras.backend.mean(x, axis=[1,2], keepdims=True))(va)
    max_pool_channel = tf.keras.layers.Lambda(lambda x: tf.keras.backend.max(x, axis=[1,2], keepdims=True))(va)
    avg_pool_channel = tf.keras.layers.Dense(input_channels//3,
                                             activation="relu", kernel_initializer='he_normal',
                                             use_bias=True,
                                             bias_initializer='zeros', name="ChannelAtt_Avg_Conv1")(avg_pool_channel)
    avg_pool_channel = tf.keras.layers.Dense(input_channels,
                                             kernel_initializer='he_normal',use_bias=True,
                                             bias_initializer='zeros', name="ChannelAtt_Avg_Conv2")(avg_pool_channel)
    max_pool_channel = tf.keras.layers.Dense(input_channels//3,
                                             activation="relu", kernel_initializer='he_normal',
                                             use_bias=True,
                                             bias_initializer='zeros', name="ChannelAtt_Max_Conv1")(max_pool_channel)
    max_pool_channel = tf.keras.layers.Dense(input_channels,
                                       kernel_initializer='he_normal',
                                       use_bias=True,
                                       bias_initializer='zeros', name="ChannelAtt_Max_Conv2")(max_pool_channel)
    channel_attention = tf.keras.layers.Add()([avg_pool_channel,max_pool_channel])
    channel_attention = tf.keras.layers.Activation("sigmoid")(channel_attention)
    channel_attention = tf.keras.layers.Multiply()([va, channel_attention])

    ## Spatial attention
    kernel_size = 7
    avg_pool_spatial = tf.keras.layers.Lambda(lambda x: tf.keras.backend.mean(x, axis=3, keepdims=True))(channel_attention)
    max_pool_spatial = tf.keras.layers.Lambda(lambda x: tf.keras.backend.max(x, axis=3, keepdims=True))(channel_attention)

    spatial_attention = tf.keras.layers.Concatenate(axis=3)([avg_pool_spatial, max_pool_spatial])
    spatial_attention = tf.keras.layers.Conv2D(filters = 1,
                                        kernel_size=kernel_size,
                                        strides=1,
                                        padding='same',
                                        activation= tf.keras.activations.sigmoid,
                                        kernel_initializer='he_normal',
                                        use_bias=False,name="SpatialAtt_Max_Conv2")(spatial_attention)
    spatial_attention = tf.keras.layers.multiply([channel_attention, spatial_attention])

    # Unify modules - Latent space
    va1 = tf.keras.layers.Conv2D(4, kernel_size=1)(va1)
    va2 = tf.keras.layers.Conv2D(4, kernel_size=1)(spatial_attention)
    va = tf.keras.layers.Add()([va1, va2])

    va = tf.keras.layers.Conv2D(4, kernel_size=5, strides=2,activation=tf.keras.activations.gelu, name="latent_output")(va)
    va = tf.keras.layers.AveragePooling2D()(va)

    va_pooling = tf.keras.layers.Flatten()(va)

    # Create module
    va_model = tf.keras.Model(inputs=inputs, outputs=va_pooling)

    # Concatenate with base model
    embedded_model = tf.keras.layers.Concatenate(name="EmbeddingLayer")([eff_model.output, va_model.output])
    #embedded_model = tf.keras.layers.Dense(512, activation="relu")(embedded_model)
    embedded_model = tf.keras.layers.Dropout(rate=0.5,name="DropoutToOutput")(embedded_model)
    outputs = tf.keras.layers.Dense(num_outputs, activation="softmax", name="Output_layer")(embedded_model)

    model = tf.keras.Model(inputs=inputs, outputs=outputs, name="FerNet")

    model.compile(loss= tf.keras.losses.CategoricalCrossentropy(label_smoothing=0.025),
                  optimizer = tf.keras.optimizers.Adam(learning_rate=1e-3),
                  metrics=["accuracy"])
```

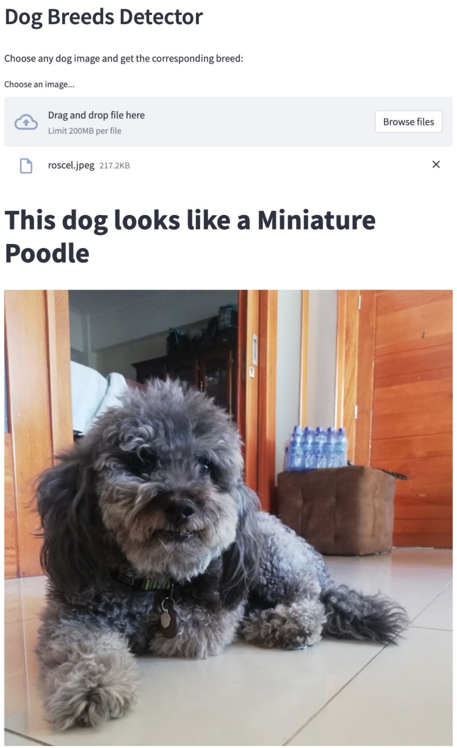
Models' Performance Results

Model	Dataset	Epochs	Best Overall Val Accuracy
EfficientNetB2 + FerNet branch	Stanford Dogs	15 + 15	89.50%
EfficientNetB2 (Base model)	Stanford Dogs	15 + 15	89.14%
Encoder only with CBAM	Stanford Dogs	15 + 15	89.07%
Encoder only with LKA	Stanford Dogs	15 + 15	88.82%
EfficientNetB2 + FerNet branch	Tsinghua Dogs	10 + 10	82.99%
EfficientNetB2 (Base model)	Tsinghua Dogs	10 + 10	82.02%

Testing FerNet (Demo)

[Dog Breed Detector App using FerNet.]

Web App Framework: **Streamlit**



Future Work

References

- EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks (ICML 2019)
- CBAM: Convolutional Block Attention Module (cs.CV 2018)
- Visual Attention Network (cs.CV 2022)