

+ Run All | Markdown ▾

Draft Session (3h:50m)

```
[1]: # This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

# import os
# for dirname, _, filenames in os.walk('/kaggle/input'):

#     for filename in filenames:
#         print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

[+ Code](#) [+ Markdown](#)

Importing Necessary Libraries

```
[3]: import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

Visualize the Data

```
[4]: import matplotlib.pyplot as plt
import os
from tensorflow.keras.preprocessing.image import load_img, img_to_array

# Directory containing the training images of cats and dogs
train_cats_dir = '/kaggle/input/catsanddogsimages/Cats and Dogs/Training/Cats 15'
train_dogs_dir = '/kaggle/input/catsanddogsimages/Cats and Dogs/Training/Dogs 15'

# Visualize some images of cats
cat_images = os.listdir(train_cats_dir)[5:] # Take the first 5 images
plt.figure(figsize=(10, 10))
for i, image in enumerate(cat_images):
    img = load_img(os.path.join(train_cats_dir, image))
    plt.subplot(1, 5, i+1)
    plt.imshow(img)
    plt.axis('off')
    plt.title('Cat')
plt.show()

# Visualize some images of dogs
dog_images = os.listdir(train_dogs_dir)[5:] # Take the first 5 images
plt.figure(figsize=(10, 10))
for i, image in enumerate(dog_images):
    img = load_img(os.path.join(train_dogs_dir, image))
    plt.subplot(1, 5, i+1)
    plt.imshow(img)
    plt.axis('off')
    plt.title('Dog')
plt.show()
```



[+ Code](#) [+ Markdown](#)

Image Manipulation and Augmentation

- using the ImageDataGenerator class from TensorFlow's Keras API

Transformations in Data Augmentation

- `Rotation_range`: This parameter specifies the range within which images can be randomly rotated. By applying random rotations, the model becomes more robust to variations in object orientations.
- `width_shift_range` and `height_shift_range`: These parameters control the range for horizontal and vertical shifts of the images, respectively. By randomly shifting the images, the model can learn to handle different object positions and improve its ability to localize objects.
- `shear_range`: This parameter applies random shear transformations to the images. Shearing distorts the shape of objects and helps the model learn to recognize objects under various perspectives.
- `zoom_range`: This parameter controls the range for random zooming of the images. Zooming in or out can introduce variations in the scale of objects, making the model more robust to changes in object sizes.
- `horizontal_flip`: This parameter enables random horizontal flipping of the images. Flipping images horizontally can help the model learn from different object orientations and improves its ability to handle mirror-image variations.

```
[5]: import os
import matplotlib.pyplot as plt
import numpy as np
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import cv2

# Directory containing the training images of cats and dogs
train_cats_dir = '/kaggle/input/catsanddogsimages/Cats and Dogs/Training/Cats 15'
train_dogs_dir = '/kaggle/input/catsanddogsimages/Cats and Dogs/Training/Dogs 15'

# Load and display an original cat image
cat_image_path = os.path.join(train_cats_dir, 'cat.1.jpg')
cat_image = cv2.imread(cat_image_path, cv2.IMREAD_GRAYSCALE) # Load as grayscale image
plt.imshow(cat_image, cmap='gray')
plt.title('Original Cat Image')
plt.axis('off')
plt.show()
```

```

# Data augmentation configuration
datagen = ImageDataGenerator(
    rotation_range=20, # Rotate images randomly within the range
    width_shift_range=0.2, # Shift images horizontally randomly
    height_shift_range=0.2, # Shift images vertically randomly
    shear_range=0.2, # Apply shear transformations
    zoom_range=0.2, # Apply zoom transformations
    horizontal_flip=True # Flip images horizontally randomly
)

# Reshape the grayscale image array to have three channels
cat_array = np.expand_dims(cat_image, axis=-1)
cat_array = np.repeat(cat_array, 3, axis=-1)

# Display augmented images of the cat
augmented_images = []
plt.figure(figsize=(10, 10))
for i in range(5):
    augmented_image = datagen.random_transform(cat_array)
    augmented_image = augmented_image[:, :, 0] # Remove the extra channel
    augmented_images.append(augmented_image)
    plt.subplot(1, 5, i+1)
    plt.imshow(augmented_image, cmap='gray')
    plt.axis('off')
    plt.title(f'Augmented Cat {i+1}')
plt.show()

```

Original Cat Image



Augmented Cat 1 Augmented Cat 2 Augmented Cat 3 Augmented Cat 4 Augmented Cat 5



CNN MODEL USING TRANSFER LEARNING

- We are using the VGG16 model as the base model, which is pre-trained on ImageNet for image classification. The base model consists of convolutional and pooling layers that extract meaningful features from the input images.
- By freezing the base model's layers, you retain the pre-trained weights and prevent them from being modified during the training process.

Reasons for Freezing the model

- Transfer learning: The base model, such as VGG16, is pre-trained on a large dataset (e.g., ImageNet) and has learned useful features that can be generalized to various computer vision tasks. By freezing the base model's layers, you can leverage these pre-trained features without altering them, which can be particularly useful when you have a limited amount of training data.
- Preventing overfitting: Freezing the base model's layers helps to prevent overfitting, especially when you have limited training data. By keeping the base model's weights fixed, you reduce the risk of the model memorizing the training data and improve its ability to generalize to unseen data.
- Efficient training: Since the base model's layers are not updated, training becomes faster because you don't have to compute gradients and update a large number of parameters. This can be especially beneficial when working with deep and complex models like VGG16.

WHY THERE IS A NEED TO INCLUDE ADDITIONAL LAYERS

- The additional layers added on top of the base model serve as the classifier for your specific task.
- The base model itself does not include a final classification layer that maps the extracted features to the specific classes you want to classify (in this case, cats and dogs).
- Therefore, we add additional layers on top of the base model to create the classifier.
- We add a Flatten layer to convert the 4D feature maps into a 1D vector. This flattening step is necessary because the subsequent layers expect a 1D input.
- Then, we add a Dense layer with 256 units and ReLU activation function to introduce non-linearity and allow the model to learn complex patterns.
- Finally, we add a Dense layer with 1 unit and a sigmoid activation function to output a probability score indicating the likelihood of the input image belonging to the positive class (e.g., a dog).

```

[6]:
import tensorflow as tf
from tensorflow.keras.applications import VGG16
from tensorflow.keras import layers

# Load the VGG16 model without the top classification layer
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(150, 150, 3))

# Freeze the base model's layers
base_model.trainable = False

# Create a new model on top of the base model
model = tf.keras.Sequential([
    base_model,
    layers.Flatten(),
    layers.Dense(256, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# Print the model summary
model.summary()

```

Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 4, 4, 512)	14714688
flatten (Flatten)	(None, 8192)	0
dense (Dense)	(None, 256)	2097408
dense_1 (Dense)	(None, 1)	257

Total params: 16,812,353
Trainable params: 2,097,665
Non-trainable params: 14,714,688

Our model architecture consists of a VGG16 base model with its pre-trained layers frozen, followed by a Flatten layer, a Dense layer with 256 units and ReLU activation, and a final Dense layer with 1 unit and sigmoid activation.

Visualizing the model

```
[7]: # Installing Visual Keras
!pip install visualkeras
import visualkeras

# Visualize the model
visualkeras.layered_view(model, scale_xy=10, legend=True)
```

Requirement already satisfied: visualkeras in /opt/conda/lib/python3.10/site-packages (0.0.2)
Requirement already satisfied: pillow>=6.2.0 in /opt/conda/lib/python3.10/site-packages (from visualkeras) (9.5.0)
Requirement already satisfied: numpy>=1.18.1 in /opt/conda/lib/python3.10/site-packages (from visualkeras) (1.23.5)
Requirement already satisfied: aggdraw>=1.3.11 in /opt/conda/lib/python3.10/site-packages (from visualkeras) (1.3.16)
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual environment instead: <https://pip.pypa.io/warnings/venv>

[7]:

Training & Testing

```
[8]: # Data augmentation configuration for testing
train_datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)

test_datagen = ImageDataGenerator() # No augmentation for testing data

# Load and augment training data
train_generator = train_datagen.flow_from_directory(
    '/kaggle/input/catsanddogsimages/Cats and Dogs/Training',
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary'
)

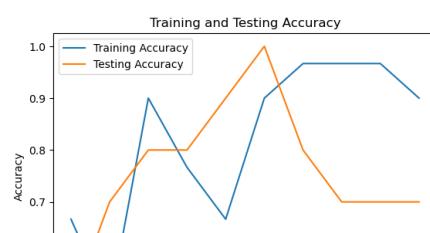
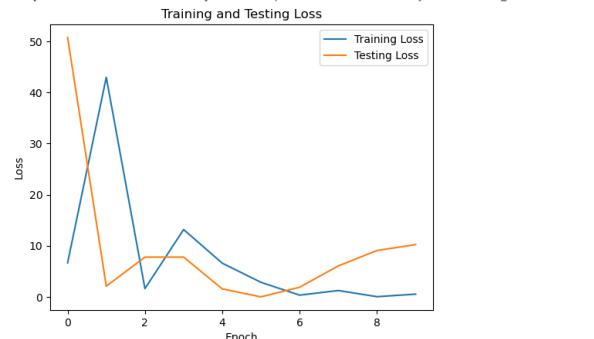
# Load testing data
test_generator = test_datagen.flow_from_directory(
    '/kaggle/input/catsanddogsimages/Cats and Dogs/Testing',
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary'
)

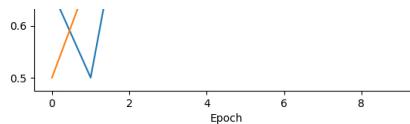
# Compile and train the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
history = model.fit(train_generator, epochs=10, validation_data = test_generator)

# Plot the combined loss curve
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Training and Testing Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Training Loss', 'Testing Loss'])
plt.show()

# Plot the combined accuracy curve
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Training and Testing Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Training Accuracy', 'Testing Accuracy'])
plt.show()
```

```
Found 30 images belonging to 2 classes.
Found 10 images belonging to 2 classes.
Epoch 1/10
1/1 [=====] - 6s 6s/step - loss: 6.6669 - accuracy: 0.6667 - val_loss: 50.7820 - val_accuracy: 0.5000
Epoch 2/10
1/1 [=====] - 5s 5s/step - loss: 42.9796 - accuracy: 0.5000 - val_loss: 2.0847 - val_accuracy: 0.7000
Epoch 3/10
1/1 [=====] - 5s 5s/step - loss: 1.6170 - accuracy: 0.9000 - val_loss: 7.7794 - val_accuracy: 0.8000
Epoch 4/10
1/1 [=====] - 5s 5s/step - loss: 13.1768 - accuracy: 0.7667 - val_loss: 7.7794 - val_accuracy: 0.8000
Epoch 5/10
1/1 [=====] - 5s 5s/step - loss: 6.5876 - accuracy: 0.6667 - val_loss: 1.5550 - val_accuracy: 0.9000
Epoch 6/10
1/1 [=====] - 5s 5s/step - loss: 2.8431 - accuracy: 0.9000 - val_loss: 0.0017 - val_accuracy: 1.0000
Epoch 7/10
1/1 [=====] - 5s 5s/step - loss: 0.3349 - accuracy: 0.9667 - val_loss: 1.8021 - val_accuracy: 0.8000
Epoch 8/10
1/1 [=====] - 5s 5s/step - loss: 1.2391 - accuracy: 0.9667 - val_loss: 6.0562 - val_accuracy: 0.7000
Epoch 9/10
1/1 [=====] - 5s 5s/step - loss: 0.0310 - accuracy: 0.9667 - val_loss: 9.0679 - val_accuracy: 0.7000
Epoch 10/10
1/1 [=====] - 5s 5s/step - loss: 0.5319 - accuracy: 0.9000 - val_loss: 10.2359 - val_accuracy: 0.7000
```





METRICS PARAMETERS

```
[9]: from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score, recall_score, f1_score
from sklearn.metrics import accuracy_score

# Predict the classes for the testing data
y_pred = model.predict(test_generator)
y_pred_classes = np.round(y_pred) # Convert probabilities to class labels (0 or 1)

# Get the true class labels
y_true = test_generator.classes

# Compute the confusion matrix
cm = confusion_matrix(y_true, y_pred_classes)
precision = precision_score(y_true, y_pred_classes)
recall = recall_score(y_true, y_pred_classes)
f1 = f1_score(y_true, y_pred_classes)
accuracy = accuracy_score(y_true, y_pred_classes)

# Print the confusion matrix
print('Confusion Matrix:', cm)
print('Precision:', precision) #The precision indicates the proportion
#of correctly predicted positive samples (cats) out of all samples predicted as positive.
print('Recall:', recall) #The recall represents the proportion of correctly
#predicted positive samples out of all actual positive samples.
print('F1 Score:', f1) #The F1 score is the harmonic mean of precision and recall,
#providing a balanced measure of the model's performance.
print('Accuracy:', accuracy)
```

1/1 [=====] - 1s 1s/step
Confusion Matrix: [[2 3]
[0 5]]
Precision: 0.625
Recall: 1.0
F1 Score: 0.7692307692307693
Accuracy: 0.7

LET'S SEE THE RESULTS !!!

```
[10]: import random

# Select a random sample of images
sample_size = 5
random_images = random.sample(test_generator.filepaths, sample_size)

# Load and preprocess the random sample images
sample_images = []
for image_path in random_images:
    img = load_img(image_path, target_size=(150, 150))
    img_array = img_to_array(img)
    img_array = img_array / 255.0
    sample_images.append(img_array)

sample_images = np.array(sample_images)

# Predict the labels for the random sample images
predictions = model.predict(sample_images)
predicted_labels = np.round(predictions).flatten()

# Get the class labels ("cat" and "dog")
class_labels = ['cat', 'dog']

# Display the random sample images with predicted labels
plt.figure(figsize=(12, 8))
for i in range(sample_size):
    plt.subplot(1, sample_size, i+1)
    plt.imshow(sample_images[i])
    plt.axis('off')
    label = class_labels[int(predicted_labels[i])]
    plt.title("Predicted: {}".format(label))
plt.show()
```

1/1 [=====] - 1s 737ms/step
Predicted: dog Predicted: dog Predicted: dog Predicted: dog Predicted: dog


Reason why the Model Works

- Transfer Learning: By using the VGG16 base model, you benefit from its pre-trained weights on ImageNet, which enables the model to learn rich and meaningful image features. This helps in capturing relevant patterns and structures in the images of cats and dogs.
- Fine-tuning: Although we froze the pre-trained layers of the base model, the added Dense layers on top allow the model to adapt and learn task-specific representations. This fine-tuning of the top layers enhances the model's ability to classify cats and dogs accurately.
- Data Augmentation: By applying data augmentation techniques during training, you increase the diversity of the training data. This helps in reducing overfitting and improving the model's generalization ability.
- Optimization: The model is compiled with the Adam optimizer, which is known for its efficiency in updating model weights. The binary cross-entropy loss function is appropriate for binary classification tasks, such as distinguishing between cats and dogs.

CAM_MODEL USING VGG16 as base model

```
[11]: import tensorflow as tf
from tensorflow.keras.applications import VGG16
from tensorflow.keras import layers

# Load the VGG16 model without the top classification layer
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(150, 150, 3))

# Freeze the base model's layers
base_model.trainable = False

# Get the output of the last convolutional layer
last_conv_layer = base_model.get_layer('block5_conv3').output

# Flatten the output of the last convolutional layer
```

```

x = layers.Flatten()(last_conv_layer)

# Add a dense layer with 256 units and ReLU activation
x = layers.Dense(256, activation='relu')(x)

# Add the final dense layer with 1 unit and sigmoid activation
output = layers.Dense(1, activation='sigmoid')(x)

# Create the new model
cam_model = tf.keras.Model(inputs=base_model.input, outputs=output)

# Compile the model
cam_model.compile(optimizer='adam',
                   loss='binary_crossentropy',
                   metrics=['accuracy'])

# Print the model summary
cam_model.summary()

```

Model: "model"

Layer (type)	Output Shape	Param #
input_2 (Inputlayer)	[None, 150, 150, 3]	0
block1_conv1 (Conv2D)	(None, 150, 150, 64)	1792
block4_conv2 (Conv2D)	(None, 150, 150, 64)	36928
block1_pool (MaxPooling2D)	(None, 75, 75, 64)	0
block2_conv1 (Conv2D)	(None, 75, 75, 128)	73856
block2_conv2 (Conv2D)	(None, 75, 75, 128)	147584
block2_pool (MaxPooling2D)	(None, 37, 37, 128)	0
block3_conv1 (Conv2D)	(None, 37, 37, 256)	295168
block3_conv2 (Conv2D)	(None, 37, 37, 256)	590080
block3_conv3 (Conv2D)	(None, 37, 37, 256)	590080
block3_pool (MaxPooling2D)	(None, 18, 18, 256)	0
block4_conv1 (Conv2D)	(None, 18, 18, 512)	1180160
block4_conv2 (Conv2D)	(None, 18, 18, 512)	2359808
block4_conv3 (Conv2D)	(None, 18, 18, 512)	2359808
block4_pool (MaxPooling2D)	(None, 9, 9, 512)	0
block5_conv1 (Conv2D)	(None, 9, 9, 512)	2359808
block5_conv2 (Conv2D)	(None, 9, 9, 512)	2359808
block5_conv3 (Conv2D)	(None, 9, 9, 512)	2359808
flatten_1 (Flatten)	(None, 41472)	0
dense_2 (Dense)	(None, 256)	10617088
dense_3 (Dense)	(None, 1)	257

Total params: 25,332,035
Trainable params: 10,617,345
Non-trainable params: 14,714,688

- Input shape: (None, 150, 150, 3)
- The model has multiple convolutional layers (Conv2D) followed by max pooling layers (MaxPooling2D) that progressively reduce the spatial dimensions of the input image while increasing the number of channels.
- The convolutional layers use various kernel sizes and numbers of filters to extract features from the input image.
- The last convolutional layer output shape is (None, 9, 9, 512).
- After the convolutional layers, there is a flattening layer (Flatten) to convert the 3D feature maps to a 1D feature vector.
- The flattened features are then passed through two fully connected layers (Dense) with 256 and 1 units, respectively.
- The final dense layer with 1 unit represents the output of the model, which is a binary classification prediction (e.g., cat or dog).

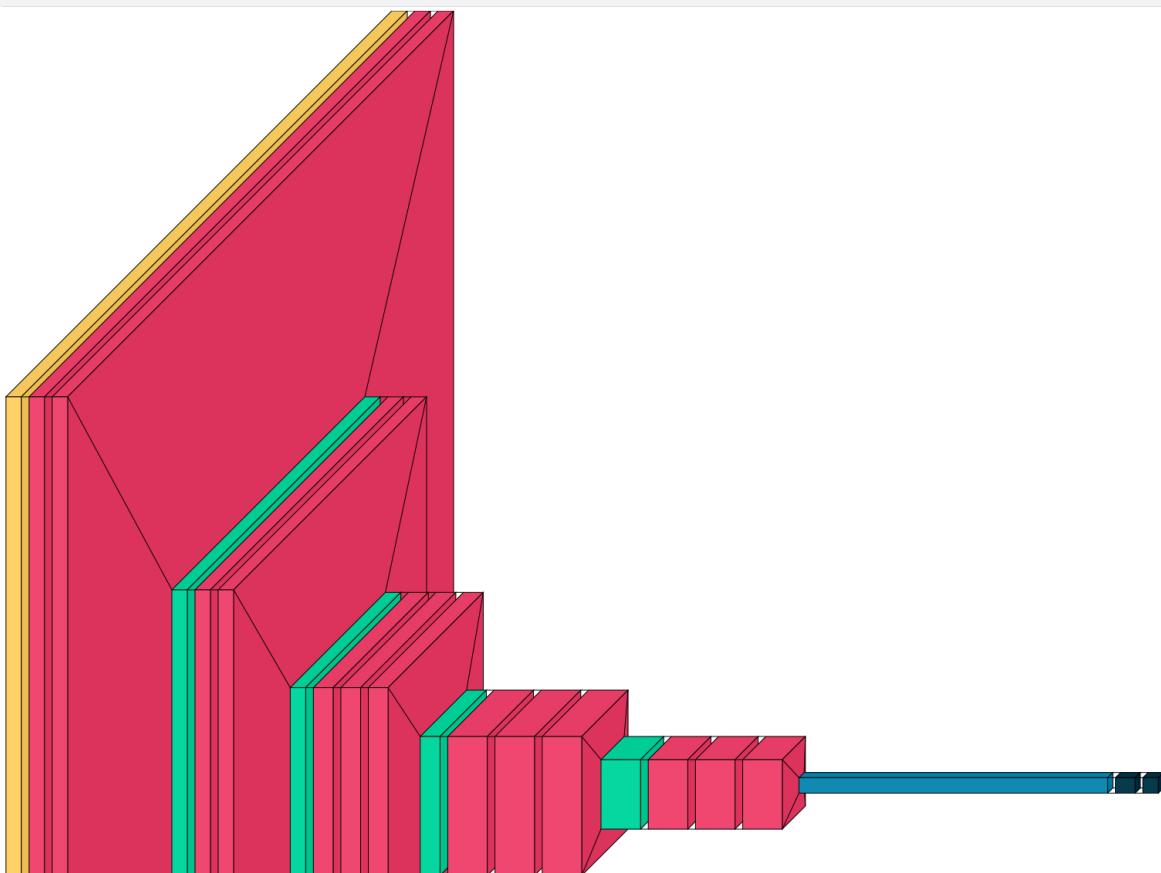
VISUALIZE THE CAM_MODEL

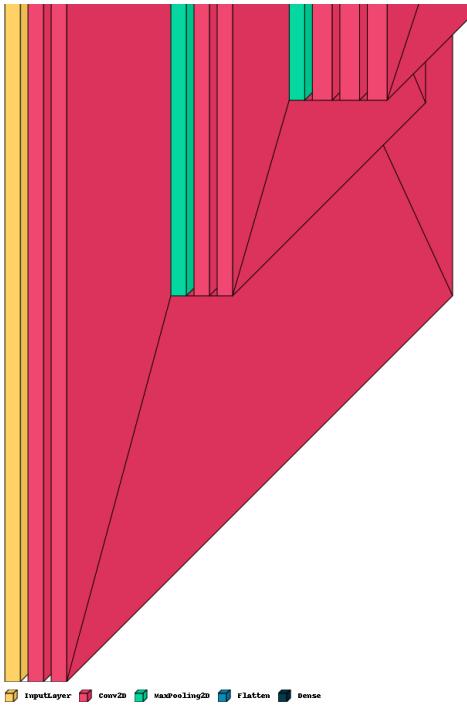
```

[12]: import visualkeras

# Visualize the model
visualkeras.layered_view(cam_model, scale_xy=10, legend=True)

```





The model has a total of 25,332,033 parameters, out of which 10,617,345 are trainable parameters, and the remaining 14,714,688 are non-trainable parameters. The trainable parameters are updated during the training process to learn from the data, while the non-trainable parameters are pre-trained weights from the VGG16 model.

TRAINING AND TESTING ON CAM_MODEL

+ Code

+ Markdown

```
[13]: # Data augmentation configuration for testing
train_datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)

test_datagen = ImageDataGenerator() # No augmentation for testing data

# Load and augment training data
train_generator = train_datagen.flow_from_directory(
    '/kaggle/input/catsanddogsimages/Cats and Dogs/Training',
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary'
)

# Load testing data
test_generator = test_datagen.flow_from_directory(
    '/kaggle/input/catsanddogsimages/Cats and Dogs/Testing',
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary'
)

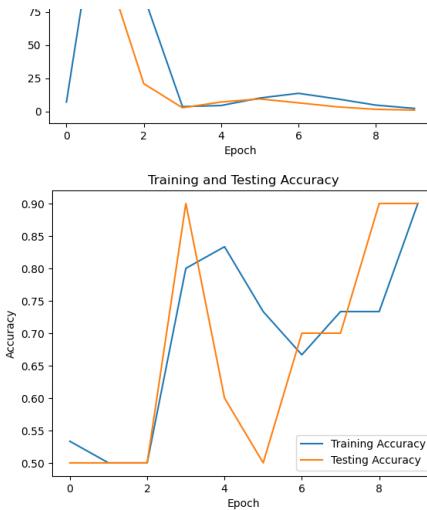
# Compile and train the model
cam_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
history = cam_model.fit(train_generator, epochs=10, validation_data = test_generator)

# Plot the combined loss curve
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Training and Testing Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Training Loss', 'Testing Loss'])
plt.show()

# Plot the combined accuracy curve
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Training and Testing Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Training Accuracy', 'Testing Accuracy'])
plt.show()
```

```
Found 30 images belonging to 2 classes.
Found 10 images belonging to 2 classes.
Epoch 1/10
1/1 [=====] - 6s/step - loss: 7.0285 - accuracy: 0.5333 - val_loss: 196.1879 - val_accuracy: 0.5000
Epoch 2/10
1/1 [=====] - 5s/step - loss: 192.7822 - accuracy: 0.5000 - val_loss: 110.5264 - val_accuracy: 0.5000
Epoch 3/10
1/1 [=====] - 5s/step - loss: 86.9389 - accuracy: 0.5000 - val_loss: 20.8069 - val_accuracy: 0.5000
Epoch 4/10
1/1 [=====] - 5s/step - loss: 3.5819 - accuracy: 0.8000 - val_loss: 2.7078 - val_accuracy: 0.9000
Epoch 5/10
1/1 [=====] - 5s/step - loss: 4.4063 - accuracy: 0.8333 - val_loss: 7.0439 - val_accuracy: 0.6000
Epoch 6/10
1/1 [=====] - 5s/step - loss: 10.0363 - accuracy: 0.7333 - val_loss: 9.3489 - val_accuracy: 0.5000
Epoch 7/10
1/1 [=====] - 5s/step - loss: 13.5741 - accuracy: 0.6667 - val_loss: 6.4250 - val_accuracy: 0.7000
Epoch 8/10
1/1 [=====] - 5s/step - loss: 9.3792 - accuracy: 0.7333 - val_loss: 3.3920 - val_accuracy: 0.7000
Epoch 9/10
1/1 [=====] - 5s/step - loss: 4.7009 - accuracy: 0.7333 - val_loss: 1.5346 - val_accuracy: 0.9000
Epoch 10/10
1/1 [=====] - 5s/step - loss: 2.2168 - accuracy: 0.9000 - val_loss: 0.9689 - val_accuracy: 0.9000
```





RESULTS OF CAM_MODEL

```
[14]:
import random

# Select a random sample of images
sample_size = 5
random_images = random.sample(test_generator.filepaths, sample_size)

# Load and preprocess the random sample images
sample_images = []
for image_path in random_images:
    img = load_img(image_path, target_size=(150, 150))
    img_array = img_to_array(img)
    img_array = img_array / 255.0
    sample_images.append(img_array)

sample_images = np.array(sample_images)

# Predict the labels for the random sample images
predictions = cam_model.predict(sample_images)
predicted_labels = np.round(predictions).flatten()

# Get the class labels ("cat" and "dog")
class_labels = ['cat', 'dog']

# Display the random sample images with predicted labels
plt.figure(figsize=(12, 8))
for i in range(sample_size):
    plt.subplot(1, sample_size, i+1)
    plt.imshow(sample_images[i])
    plt.axis('off')
    label = class_labels[int(predicted_labels[i])]
    plt.title('Predicted: {}'.format(label))
plt.show()
```



Visualizing CAM using Heatmap

- It loads and preprocesses the image using the same steps as before.
- It creates a submodel that extracts the output of the last convolutional layer (last_conv_layer_name) and the model's predicted class.
- It calculates the gradients of the predicted class with respect to the last convolutional layer output.
- It applies the CAM formula to obtain the heatmap.
- The heatmap is normalized and resized to match the image size.
- The heatmap is converted to a numpy array and the colormaps are applied to it.
- The original image is loaded using OpenCV, and its dimensions are retrieved.
- The heatmap is resized to match the dimensions of the input image.
- The heatmap is overlaid on the original image using an additive blending technique.
- The original image, heatmap, and overlay are displayed in the corresponding subplots.

```
[32]:
import cv2
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.applications.vgg16 import preprocess_input

def apply_cam_to_images(image_paths, model):
    fig, axes = plt.subplots(len(image_paths), 3, figsize=(12, 4 * len(image_paths)))

    for i, image_path in enumerate(image_paths):
        # Load and preprocess the image
        img = tf.keras.preprocessing.image.load_img(image_path, target_size=(150, 150))
        img_array = tf.keras.preprocessing.image.img_to_array(img)
        img_array = img_array / 255.0
        img_array = np.expand_dims(img_array, axis=0)

        # Create a submodel that extracts the last convolutional layer output and the model's predicted class
        last_conv_layer_name = 'block5_conv3'
        submodel = tf.keras.Model(inputs=model.input, outputs=[model.get_layer(last_conv_layer_name).output, model.output])

        # Calculate the gradients of the predicted class with respect to the last convolutional layer output
        with tf.GradientTape() as tape:
            conv_outputs, predictions = submodel(img_array)
            predicted_class = tf.argmax(predictions, axis=-1)[0] # Extract the scalar value
            class_output = predictions[:, predicted_class]

            grads = tape.gradient(class_output, conv_outputs)
            pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))

        # Multiply the gradients by the last convolutional layer output and take the mean
        heatmap = tf.reduce_mean(tf.multiply(pooled_grads, conv_outputs), axis=-1)
```

```

# Normalize the heatmap
heatmap = tf.maximum(heatmap, 0)
heatmap /= tf.reduce_max(heatmap)

# Resize the heatmap to match the image size
heatmap = tf.image.resize(heatmap, (img_array.shape[1], img_array.shape[2]))

# Convert the heatmap to a numpy array
heatmap = heatmap.numpy()[0]

# Apply the colormap to the heatmap
heatmap = cv2.applyColorMap(np.uint8(255 * heatmap), cv2.COLORMAP_JET)

# Get the dimensions of the input image
img = cv2.imread(image_path)
img_height, img_width, _ = img.shape

# Resize the heatmap to match the dimensions of the input image
resized_heatmap = cv2.resize(heatmap, (img_width, img_height))
resized_heatmap = cv2.cvtColor(resized_heatmap, cv2.COLOR_RGB2BGR)

# Overlay the heatmap on the original image
overlay = cv2.addWeighted(img, 0.5, resized_heatmap, 0.5, 0)

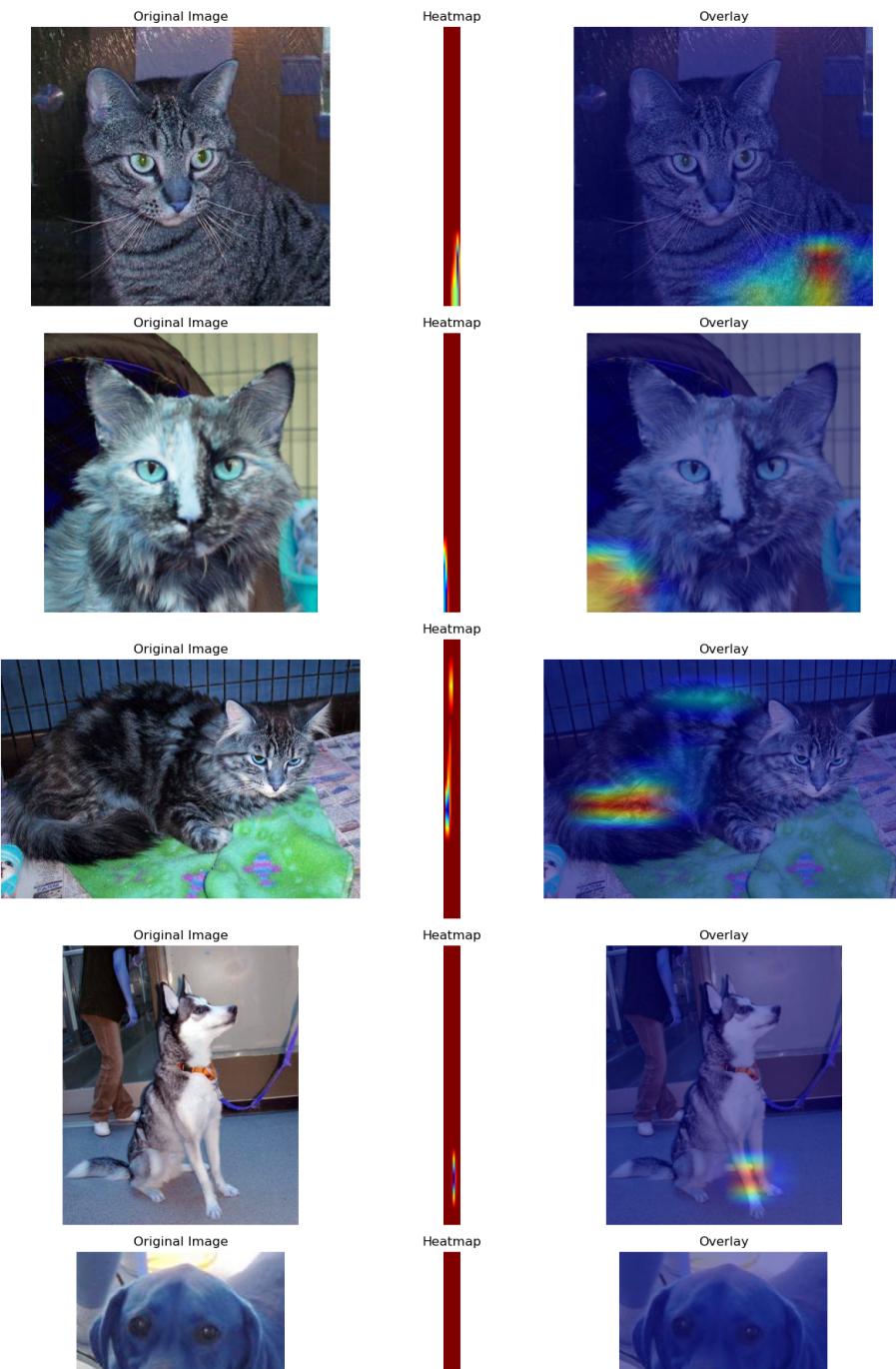
# Display the original image, heatmap, and overlay
axes[1, 0].imshow(img)
axes[1, 0].set_title('Original Image')
axes[1, 0].axis('off')
axes[1, 1].imshow(heatmap)
axes[1, 1].set_title('Heatmap')
axes[1, 1].axis('off')
axes[1, 2].imshow(overlay)
axes[1, 2].set_title('Overlay')
axes[1, 2].axis('off')

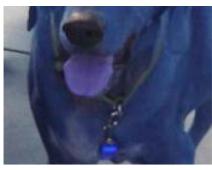
plt.tight_layout()
plt.show()

image_paths = [
    '/kaggle/input/catsanddogsimages/Cats and Dogs/Training/Cats 15/cat.1.jpg',
    '/kaggle/input/catsanddogsimages/Cats and Dogs/Training/Cats 15/cat.10.jpg',
    '/kaggle/input/catsanddogsimages/Cats and Dogs/Training/Cats 15/cat.1005.jpg',
    '/kaggle/input/catsanddogsimages/Cats and Dogs/Training/Dogs 15/dog.4001.jpg',
    '/kaggle/input/catsanddogsimages/Cats and Dogs/Training/Dogs 15/dog.4014.jpg'
]

```

```
apply_cam_to_images(image_paths, cam_model)
```





□

