```python
import os
import numpy as np
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt
import plotly.express as px

from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.layers import Conv2D,Add,MaxPooling2D, Dense, BatchNormalization,Input,Flatten, Dropout,GlobalMax
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam,RMSprop
from tensorflow.keras.callbacks import LearningRateScheduler
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```python
#unzip files
!unzip -q Pets.zip
!ls
!ls test
```

```
    replace test/cats/cat_1.jpg? [y]es, [n]o, [A]ll, [N]one, [r]ename: N
    model.png  Pets.zip  sample_data  test  train
    cats  dogs
```

```python
#split data
image_size = (180, 180)
batch_size = 128

train_ds, val_ds = tf.keras.utils.image_dataset_from_directory(
    "train",
    validation_split=0.2,
    subset="both",
    seed=1337,
    image_size=image_size,
    batch_size=batch_size,
)
```

```
    Found 557 files belonging to 2 classes.
    Using 446 files for training.
    Using 111 files for validation.
```

```python
#show sample of data labeling 1 for dog 0 for cat
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(int(labels[i]))
        plt.axis("off")
```
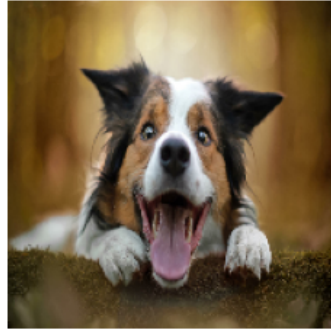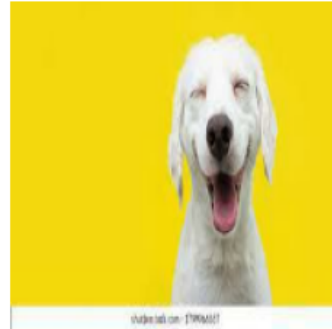
```
#augment data
data_augmentation = keras.Sequential(
    [
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.1),
    ]
)
```



```
#show sample of augmented data
plt.figure(figsize=(10, 10))
for images, _ in train_ds.take(1):
    for i in range(9):
        augmented_images = data_augmentation(images)
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(augmented_images[0].numpy().astype("uint8"))
        plt.axis("off")
```
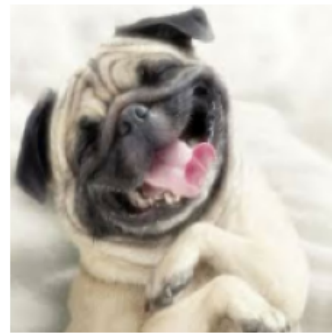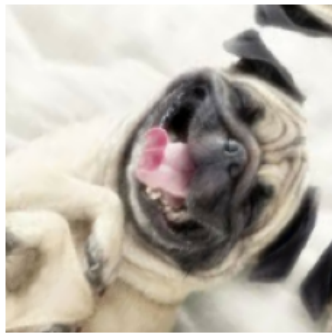
```python
#augment data
augmented_train_ds = train_ds.map(
    lambda x, y: (data_augmentation(x, training=True), y))
```



```python
#get data
train_ds = train_ds.map(
    lambda img, label: (data_augmentation(img), label),
    num_parallel_calls=tf.data.AUTOTUNE,
)
train_ds = train_ds.prefetch(tf.data.AUTOTUNE)
val_ds = val_ds.prefetch(tf.data.AUTOTUNE)
```



```python
#define architecture of model
def make_model(input_shape, num_classes):
    inputs = keras.Input(shape=input_shape)

    x = layers.Rescaling(1.0 / 255)(inputs)
    x = layers.Conv2D(128, 3, strides=2, padding="same")(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation("relu")(x)

    previous_block_activation = x

    for size in [256, 512, 728]:
        x = layers.Activation("relu")(x)
        x = layers.SeparableConv2D(size, 3, padding="same")(x)
        x = layers.BatchNormalization()(x)

        x = layers.Activation("relu")(x)
        x = layers.SeparableConv2D(size, 3, padding="same")(x)
        x = layers.BatchNormalization()(x)

        x = layers.MaxPooling2D(3, strides=2, padding="same")(x)

        residual = layers.Conv2D(size, 1, strides=2, padding="same")(
            previous_block_activation
        )
        x = layers.add([x, residual])
        previous_block_activation = x  # Set aside next residual

    x = layers.SeparableConv2D(1024, 3, padding="same")(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation("relu")(x)

    x = layers.GlobalAveragePooling2D()(x)
    if num_classes == 2:
        activation = "sigmoid"
        units = 1
    else:
        activation = "softmax"
        units = num_classes
```

```
    x = layers.Dropout(0.5)(x)
    outputs = layers.Dense(units, activation=activation)(x)
    return keras.Model(inputs, outputs)


model = make_model(input_shape=image_size + (3,), num_classes=2)
keras.utils.plot_model(model, show_shapes=True)
```

separable_conv2d_3	input:	(None, 45, 45, 512)
SeparableConv2D	output:	(None, 45, 45, 512)

batch_normalization_4	input:	(None, 45, 45, 512)
BatchNormalization	output:	(None, 45, 45, 512)

max_pooling2d_9	input:	(None, 45, 45, 512)
MaxPooling2D	output:	(None, 23, 23, 512)

add_1	input:	[(None, 23, 23, 512), (None, 23, 23, 512)]
Add	output:	(None, 23, 23, 512)

activation_5	input:	(None, 23, 23, 512)
Activation	output:	(None, 23, 23, 512)

separable_conv2d_4	input:	(None, 23, 23, 512)
SeparableConv2D	output:	(None, 23, 23, 728)

conv2d_11	input:	(None, 23, 23, 512)
Conv2D	output:	(None, 12, 12, 728)

batch_normalization_5	input:	(None, 23, 23, 728)
BatchNormalization	output:	(None, 23, 23, 728)

activation_6	input:	(None, 23, 23, 728)
Activation	output:	(None, 23, 23, 728)

separable_conv2d_5	input:	(None, 23, 23, 728)
SeparableConv2D	output:	(None, 23, 23, 728)

batch_normalization_6	input:	(None, 23, 23, 728)
BatchNormalization	output:	(None, 23, 23, 728)

max_pooling2d_10	input:	(None, 23, 23, 728)
MaxPooling2D	output:	(None, 12, 12, 728)

add_2	input:	[(None, 12, 12, 728), (None, 12, 12, 728)]
Add	output:	(None, 12, 12, 728)

```python
#evaluate using CNN
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, BatchNormalization, Input, Flatten, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam, RMSprop
from tensorflow.keras.preprocessing.image import ImageDataGenerator

model = keras.Sequential([
    layers.experimental.preprocessing.Rescaling(1./255, input_shape=(180, 180, 3)),
    layers.Conv2D(32, 3, activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(64, 3, activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(128, 3, activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(256, 3, activation='relu'),
    layers.MaxPooling2D(),
    layers.Flatten(),
    layers.Dense(512, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(1, activation='sigmoid')
])
```

| Dropout | output | (None, 1024) |

```python
model.compile(
    optimizer=keras.optimizers.Adam(1e-3),
    loss="binary_crossentropy",
    metrics=["accuracy"],
)

history = model.fit(
    train_ds,
    epochs=10,
    callbacks=[keras.callbacks.ModelCheckpoint("save_at_{epoch}.keras"),],
    validation_data=val_ds,
)
```

```
=======================] - 50s 11s/step - loss: 1.1292 - accuracy: 0.4910 - val_loss: 0.7149 - val_accuracy: 0.48(

=======================] - 52s 11s/step - loss: 0.7032 - accuracy: 0.5045 - val_loss: 0.6938 - val_accuracy: 0.48(

=======================] - 47s 11s/step - loss: 0.6934 - accuracy: 0.5067 - val_loss: 0.6921 - val_accuracy: 0.51:

=======================] - 46s 11s/step - loss: 0.6934 - accuracy: 0.5000 - val_loss: 0.6932 - val_accuracy: 0.49!

=======================] - 46s 11s/step - loss: 0.6923 - accuracy: 0.5112 - val_loss: 0.6918 - val_accuracy: 0.49!

=======================] - 47s 11s/step - loss: 0.6866 - accuracy: 0.5785 - val_loss: 0.6926 - val_accuracy: 0.50

=======================] - 44s 10s/step - loss: 0.6782 - accuracy: 0.5650 - val_loss: 0.7117 - val_accuracy: 0.51:

=======================] - 47s 11s/step - loss: 0.6957 - accuracy: 0.5157 - val_loss: 0.6868 - val_accuracy: 0.51:

=======================] - 53s 10s/step - loss: 0.6793 - accuracy: 0.5628 - val_loss: 0.6738 - val_accuracy: 0.60:

=======================] - 46s 10s/step - loss: 0.6568 - accuracy: 0.6413 - val_loss: 0.6789 - val_accuracy: 0.58!
```

```python
#Train model

model.compile(
    optimizer=keras.optimizers.Adam(1e-3),
    loss="binary_crossentropy",
    metrics=["accuracy"],
)

history = model.fit(
    train_ds,
    epochs= 5,
    callbacks=[LearningRateScheduler(lambda epoch: 1e-3 * (0.8 ** epoch))]
```