Pet breed detection with ResNet50

Data import and discovery

Library and data import

In [1]:
```python
import tensorflow as tf
import tensorflow_datasets as tfds
from tensorflow import keras
from tensorflow.keras import layers, models
from tensorflow.keras.applications.resnet50 import preprocess_input, ResNet50

import numpy as np
from numpy import round, sqrt, random
import matplotlib.pyplot as plt



dataset, info = tfds.load('oxford_iiit_pet',split=['train[:80%]', 'train[80%:]','test'], as_supervised=True, wi
train_set_raw, valid_set_raw, test_set_raw = dataset

# Create a dictionary of numerical label - breed key-value pairs
labels = info.features['label'].names                          # Name of the breed
label_dict = {i: breed for i,breed in enumerate(labels)}    # Dict comprehension to create key-value pairs of nu
print(label_dict)

# Find out about the pixel value range
for image,_ in train_set_raw.take(1):
    print(f"Pixel value range: [{tf.reduce_min(image).numpy()}, {tf.reduce_max(image).numpy()}]")

# Number of classes
num_classes = info.features['label'].num_classes
print(f"There are {num_classes} classes of dogs and cats in this dataset")
```

```
{0: 'Abyssinian', 1: 'american_bulldog', 2: 'american_pit_bull_terrier', 3: 'basset_hound', 4: 'beagle', 5: 'Ben
gal', 6: 'Birman', 7: 'Bombay', 8: 'boxer', 9: 'British_Shorthair', 10: 'chihuahua', 11: 'Egyptian_Mau', 12: 'en
glish_cocker_spaniel', 13: 'english_setter', 14: 'german_shorthaired', 15: 'great_pyrenees', 16: 'havanese', 17:
'japanese_chin', 18: 'keeshond', 19: 'leonberger', 20: 'Maine_Coon', 21: 'miniature_pinscher', 22: 'newfoundland
', 23: 'Persian', 24: 'pomeranian', 25: 'pug', 26: 'Ragdoll', 27: 'Russian_Blue', 28: 'saint_bernard', 29: 'samo
yed', 30: 'scottish_terrier', 31: 'shiba_inu', 32: 'Siamese', 33: 'Sphynx', 34: 'staffordshire_bull_terrier', 35
: 'wheaten_terrier', 36: 'yorkshire_terrier'}
Pixel value range: [0, 255]
There are 37 classes of dogs and cats in this dataset
```

Input data visualization

In this section, I will visualize the images and labels from the dataset

In [2]:
```python
num_samples = 9
num_rows = int(round(sqrt(num_samples))); num_cols = int(num_samples/num_rows)
index = 1
plt.figure(figsize=(num_rows*3,num_cols*3))

# Display some images from the raw train set
for image, label in train_set_raw.take(num_samples):
    # print(image)
    plt.subplot(num_rows,num_cols,index)
    plt.imshow(image.numpy().astype("uint8"))
    plt.title(f"Label: {label_dict[label.numpy()]}")
    plt.axis("off")
    index += 1
plt.show()
plt.tight_layout
```

Out[2]:
```
<function matplotlib.pyplot.tight_layout(*, pad: 'float' = 1.08, h_pad: 'float | None' = None, w_pad: 'float | None' = None, rect: 'tuple[float, float, float, float] | None' = None) -> 'None'>
```

Examine the image dimensions

As can be seen from the visualization above, each iamge has a unique size. I ahve created the function below to determine the minimum height and width resolution of the images in the train, validation, and test set

In [3]:
```python
def find_min_resolution(dataset):

    min_height = float('inf')
    min_width = float('inf')
    for image, _ in dataset:
        height,width,_ = image.shape

        min_height = min(min_height,height)
        min_width = min(min_width,width)

    print(f"The minimum height resolution: {min_height}\n"
          f"The minimum width resolution: {min_width}")

find_min_resolution(train_set_raw)
find_min_resolution(valid_set_raw)
find_min_resolution(test_set_raw)
```

```
The minimum height resolution: 108
The minimum width resolution: 114
The minimum height resolution: 112
The minimum width resolution: 150
The minimum height resolution: 103
The minimum width resolution: 137
```

Image resizing

So the minimum height resolution is 103 and the minimum width resolution is 114. I will resize the images to have dimension of (224,224) for the pretrained pre-trained ResNet50 network. Normally, I would include the iamge preprocessing step inside the final model. However, given the variety of image dimensions, I will have to resize the images first

In [4]:
```
def preprocess_image(image,target_size = (96, 96),
                     display=False,
                     pad=True):
    '''This function resize an image while keeping the aspect ratio such that the shorter side is 96 pixels (by

    if pad:     # Resize with padding (shrink the image while preserving aspect ratio and fill void with black)
        image = tf.image.resize_with_pad(image, target_size[0],target_size[1])
    else:       # Resize without padding - stretch or shrink the image to the desired target size
        image = tf.image.resize(image, target_size)

    return image

def preprocess_dataset(dataset, target_size = (96,96),display=False,pad=True):
    '''This function applies the preprocess_image() on the images in a dataset to resize the iamges to the targe
    return dataset.map(lambda image, label: (preprocess_image(image,target_size,display,pad), label))

# The desired size for the processed images
TARGET_SIZE = (224,224)
train_set_processed = preprocess_dataset(train_set_raw,target_size=TARGET_SIZE)
valid_set_processed = preprocess_dataset(valid_set_raw,target_size=TARGET_SIZE)
test_set_processed = preprocess_dataset(test_set_raw,target_size=TARGET_SIZE)

train_set_processed_display = preprocess_dataset(train_set_raw,target_size=TARGET_SIZE,display=True)

find_min_resolution(train_set_processed_display)
```

```
The minimum height resolution: 224
The minimum width resolution: 224
```

In [5]:
```
num_samples = 9
num_rows = int(round(sqrt(num_samples))); num_cols = int(num_samples/num_rows)
index = 1
plt.figure(figsize=(num_rows*3,num_cols*3))

# Display some images from the raw train set
for image, label in train_set_processed_display.take(num_samples):
    # # Examine the range of pixel value
    # print("Pixel value range:", tf.reduce_min(image).numpy(), "to", tf.reduce_max(image).numpy())

    plt.subplot(num_rows,num_cols,index)
    plt.imshow(image.numpy().astype("uint8"))
    plt.title(f"Label: {label_dict[label.numpy()]}")
    plt.axis("off")
    index += 1
plt.show()
plt.tight_layout
```

Out[5]:
```
<function matplotlib.pyplot.tight_layout(*, pad: 'float' = 1.08, h_pad: 'float | None' = None, w_pad: 'float | None' = None, rect: 'tuple[float, float, float, float] | None' = None) -> 'None'>
```

Dataset => Array Block

In this code block, I define a function to extract the image and label data from any of the three datasets. This make it easier to work with the data in the form of numpy arrays instead of PrefetchDataset objects when importing the dataset from tensorflow.

Since the dataset has images of different dimensions, this function will also resize the images to a

In [6]:
```
def get_nparray_dataset(dataset):
    image_list = []
    label_list = []

    for image,label in dataset:
        image_list.append(image.numpy())
        label_list.append(label.numpy())

    image_list_npararay = np.array(image_list)
    label_list_nparray = np.array(label_list)

    return image_list_npararay, label_list_nparray
```

In [7]:
```
train_image_array, train_label_array = get_nparray_dataset(train_set_processed)
valid_image_array, valid_label_array = get_nparray_dataset(valid_set_processed)
test_image_array, test_label_array = get_nparray_dataset(test_set_processed)

print(train_image_array.shape, train_label_array.shape)
print(valid_image_array.shape, valid_label_array.shape)
print(test_image_array.shape, test_label_array.shape)

num_train = len(train_label_array)
num_valid = len(valid_label_array)
num_test = len(test_image_array)
```

```
(2944, 224, 224, 3) (2944,)
(736, 224, 224, 3) (736,)
(3669, 224, 224, 3) (3669,)
```

ResNet50-based model

Base model construction

In [8]:
```
base_model = ResNet50(weights='imagenet',
                    #   input_shape=TARGET_SIZE+(3,),
                      include_top=False)

# Freeze the weights of the model
base_model.trainable = False

# # Investigate the structure of the base model and make sure that the weights are frozen
# base_model.summary()
```

Full model architecture

In [9]:
```
model_resnet = models.Sequential([
    layers.Input(shape=TARGET_SIZE + (3,)),
    layers.Lambda(preprocess_input),
    base_model,
    layers.GlobalAveragePooling2D(),
    layers.BatchNormalization(),
    layers.Dropout(0.5),
    layers.Dense(num_classes, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(num_classes, activation='softmax')  # Output layer for pet breeds
])

initial_weights = model_resnet.get_weights()

model_resnet.summary()
```

```
WARNING:tensorflow:From c:\mnguyen\TME_6015\.venv\Lib\site-packages\keras\src\backend\tensorflow\core.py:204: Th
e name tf.placeholder is deprecated. Please use tf.compat.v1.placeholder instead.
```


```
WARNING:tensorflow:From c:\mnguyen\TME_6015\.venv\Lib\site-packages\keras\src\backend\tensorflow\core.py:204: Th
e name tf.placeholder is deprecated. Please use tf.compat.v1.placeholder instead.
```


Model: "sequential"

Layer (type)	Output Shape	Param #
lambda (Lambda)	(None, 224, 224, 3)	0
resnet50 (Functional)	(None, 7, 7, 2048)	23,587,712
global_average_pooling2d (GlobalAveragePooling2D)	(None, 2048)	0
batch_normalization (BatchNormalization)	(None, 2048)	8,192
dropout (Dropout)	(None, 2048)	0
dense (Dense)	(None, 37)	75,813
dropout_1 (Dropout)	(None, 37)	0
dense_1 (Dense)	(None, 37)	1,406

Total params: 23,673,123 (90.31 MB)

Trainable params: 81,315 (317.64 KB)

Non-trainable params: 23,591,808 (90.00 MB)

Model Training

In [10]:
```
model_resnet.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.005),
              loss=tf.keras.losses.SparseCategoricalCrossentropy,
              metrics=["accuracy"])
loss0, acc0 = model_resnet.evaluate(test_image_array,test_label_array)

print("initial loss: {:.2f}".format(loss0))
print("initial accuracy: {:.2f}".format(acc0))
```

```
115/115 ━━━━━━━━━━━━━━━━━━━━ 41s 346ms/step - accuracy: 0.0306 - loss: 3.8299
initial loss: 3.80
initial accuracy: 0.03
```

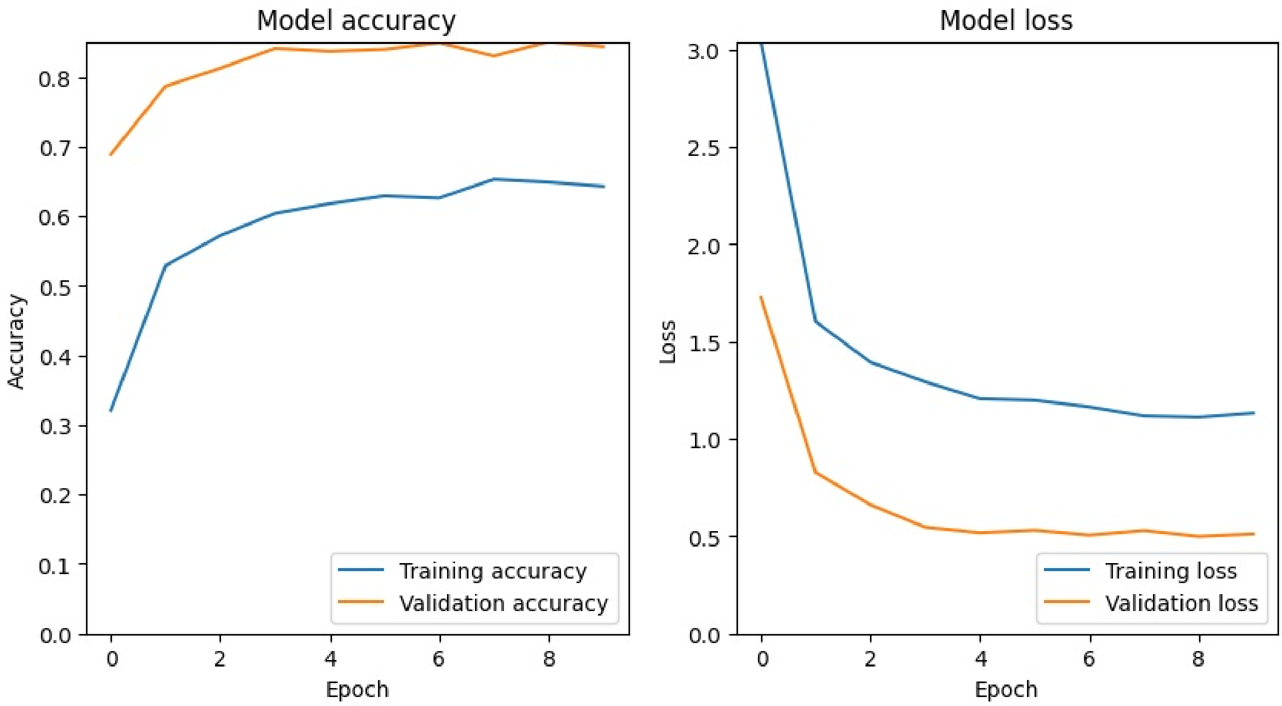
In [42]:
```
# Train the model with the base layers frozen
initial_epochs = 10
model_resnet.set_weights(initial_weights)
history_resnet = model_resnet.fit(train_image_array,train_label_array,
                                  validation_data=(valid_image_array,valid_label_array),
                                  epochs=initial_epochs)
```

```
Epoch 1/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 40s 438ms/step - accuracy: 0.2098 - loss: 3.8667 - val_accuracy: 0.6889 - val_loss: 1
.7264
Epoch 2/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 40s 438ms/step - accuracy: 0.5103 - loss: 1.6517 - val_accuracy: 0.7867 - val_loss: 0
.8276
Epoch 3/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 40s 438ms/step - accuracy: 0.5683 - loss: 1.4143 - val_accuracy: 0.8125 - val_loss: 0
.6622
Epoch 4/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 41s 446ms/step - accuracy: 0.5948 - loss: 1.3091 - val_accuracy: 0.8410 - val_loss: 0
.5467
Epoch 5/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 41s 447ms/step - accuracy: 0.6039 - loss: 1.2261 - val_accuracy: 0.8370 - val_loss: 0
.5180
Epoch 6/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 40s 434ms/step - accuracy: 0.6373 - loss: 1.1347 - val_accuracy: 0.8397 - val_loss: 0
.5309
Epoch 7/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 41s 444ms/step - accuracy: 0.6220 - loss: 1.1613 - val_accuracy: 0.8492 - val_loss: 0
.5059
Epoch 8/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 41s 444ms/step - accuracy: 0.6413 - loss: 1.1203 - val_accuracy: 0.8302 - val_loss: 0
.5293
Epoch 9/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 39s 427ms/step - accuracy: 0.6550 - loss: 1.0809 - val_accuracy: 0.8505 - val_loss: 0
.4990
Epoch 10/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 39s 430ms/step - accuracy: 0.6436 - loss: 1.1059 - val_accuracy: 0.8438 - val_loss: 0
.5118
```

In [43]:
```
def plot_performance(history, learning_rate=None, batch_size=None, finetune_epochs=None):
  plt.figure(figsize=(10,5))

  # history_data = history.history

  # Determine whether history is keras history or a dictionary to appropriately extract the history data
  if isinstance(history, keras.callbacks.History):
    history_data = history.history        # Extract the history dictionary
  else:
    history_data = history                # Assume it's already a dictionary

  # Accuracy of model training and validation vs training epoch
  plt.subplot(1,2,1)
  ylim_acc = [0, max(max(history_data['accuracy']),max(history_data['val_accuracy']))]
  plt.plot(history_data['accuracy'], label = 'Training accuracy')
  plt.plot(history_data['val_accuracy'], label = 'Validation accuracy')
  plt.ylim(ylim_acc)
  if finetune_epochs:
    plt.plot([finetune_epochs-1, finetune_epochs-1],plt.ylim(), label = 'Fine tuning')
  else:
    pass

  if learning_rate and batch_size:
    plt.title(f'Model accuracy \n lr = {learning_rate}, batch size = {batch_size}')
  else: plt.title('Model accuracy')

  plt.ylabel('Accuracy')
  plt.xlabel('Epoch')
  plt.legend(loc='lower right')

  # Loss during model training and validation
  plt.subplot(1,2,2)
  ylim_loss = [0, max(max(history_data['loss']),max(history_data['val_loss']))]
  # print(len(history_data['loss']))
  plt.plot(history_data['loss'], label = 'Training loss')
  plt.plot(history_data['val_loss'], label = 'Validation loss')
  plt.ylim(ylim_loss)
  if finetune_epochs:
    plt.plot([finetune_epochs-1, finetune_epochs-1],plt.ylim(), label = 'Fine tuning')
  else:
    pass

  if learning_rate and batch_size:
    plt.title(f'Model loss \n lr = {learning_rate}, batch size = {batch_size}')
  else: plt.title('Model loss')
  plt.ylabel('Loss')
```

```
    plt.xlabel('Epoch')
    plt.legend(loc='lower right')
    plt.show()

    print(f"The model has a training accuracy of {history_data['accuracy'][-1]*100:.2f}%\n"
        f"The model has a validation accuracy of {history_data['val_accuracy'][-1]*100:.2f}%")
    return
```

In [44]:
```
plot_performance(history_resnet)
```




```
The model has a training accuracy of 64.27%
The model has a validation accuracy of 84.38%
```

Model Evaluation

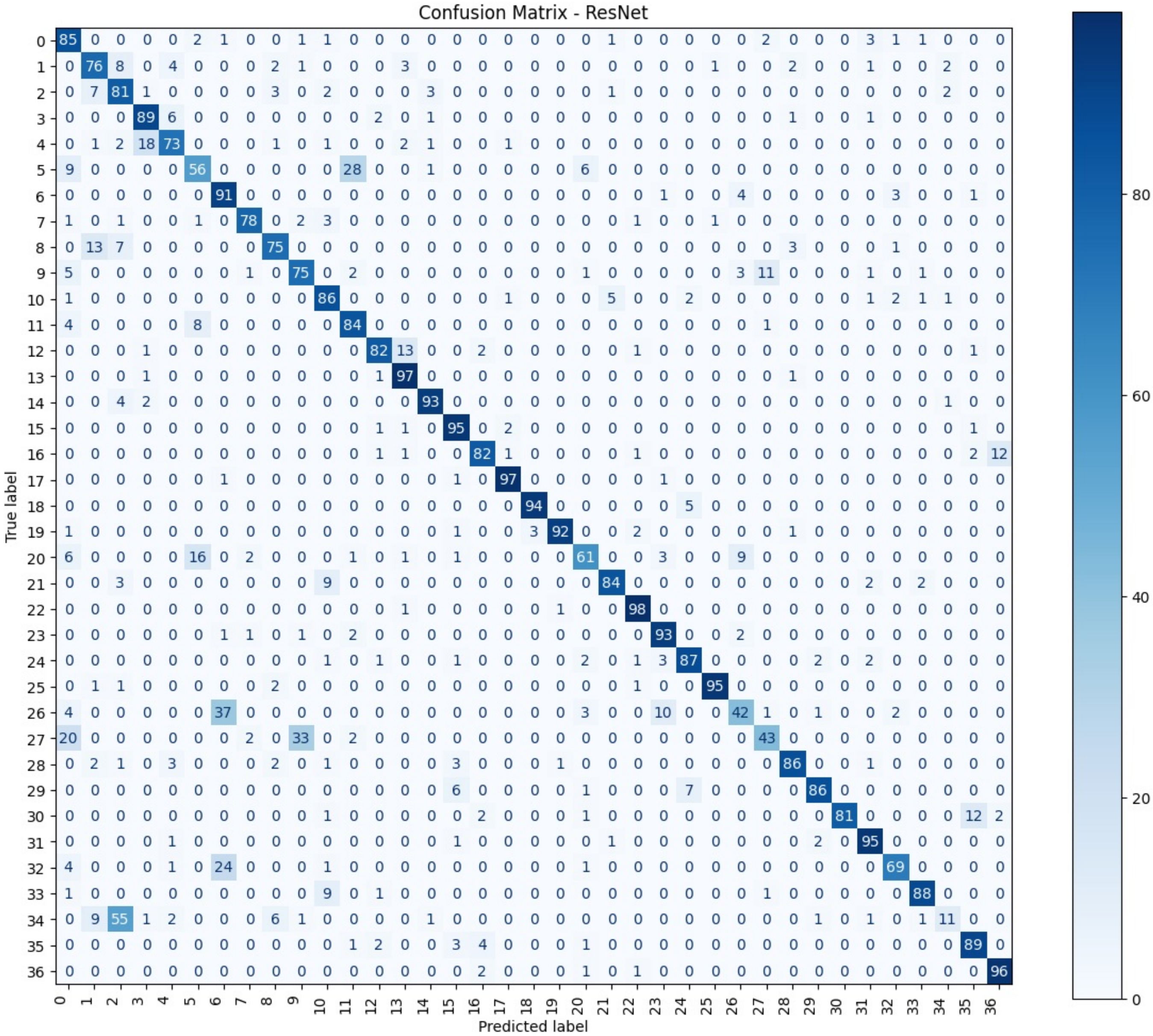
In [14]:
```
test_loss, test_acc = model_resnet.evaluate(test_image_array,test_label_array)
print(f"Test accuracy: {test_acc}\n"
      f"Test loss: {test_loss}")
```

```
115/115 ━━━━━━━━━━━━━━━━━━━━ 41s 353ms/step - accuracy: 0.8212 - loss: 0.5542
Test accuracy: 0.8135731816291809
Test loss: 0.5614109039306641
```

In [15]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

prediction_array = np.argmax(model_resnet.predict(test_image_array), axis=1)
```

```
115/115 ━━━━━━━━━━━━━━━━━━━━ 42s 355ms/step
```

In [34]:
```
cm = confusion_matrix(test_label_array, prediction_array)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap=plt.cm.Blues)  # You can change the color map as desired
fig = disp.ax_.get_figure()
fig.set_figwidth(12); fig.set_figheight(10)
plt.title("Confusion Matrix - ResNet")
plt.xticks(rotation=90, ha='right')  # Rotate x labels for better readability
plt.yticks(rotation=0)  # Keep y labels horizontal
plt.tight_layout()  # Adjust layout to make room for rotated labels
plt.show()

label_dict
```


Confusion Matrix - ResNet
True label
Predicted label
0
20
40
60
80

Out[34]:
```
{0: 'Abyssinian',
 1: 'american_bulldog',
 2: 'american_pit_bull_terrier',
 3: 'basset_hound',
 4: 'beagle',
 5: 'Bengal',
 6: 'Birman',
 7: 'Bombay',
 8: 'boxer',
 9: 'British_Shorthair',
 10: 'chihuahua',
 11: 'Egyptian_Mau',
 12: 'english_cocker_spaniel',
 13: 'english_setter',
 14: 'german_shorthaired',
 15: 'great_pyrenees',
 16: 'havanese',
 17: 'japanese_chin',
 18: 'keeshond',
 19: 'leonberger',
 20: 'Maine_Coon',
 21: 'miniature_pinscher',
 22: 'newfoundland',
 23: 'Persian',
 24: 'pomeranian',
 25: 'pug',
 26: 'Ragdoll',
 27: 'Russian_Blue',
 28: 'saint_bernard',
 29: 'samoyed',
 30: 'scottish_terrier',
 31: 'shiba_inu',
 32: 'Siamese',
 33: 'Sphynx',
 34: 'staffordshire_bull_terrier',
 35: 'wheaten_terrier',
 36: 'yorkshire_terrier'}
```

Model Prediction and Visualization

In [ ]:
```
# Sample random images and their indices
num_samples = 9                                                                        # number of samples
num_rows = int(round(sqrt(num_samples))); num_cols = int(num_samples/num_rows)      # number of rows and column
rand = random.randint(num_test,size = (num_samples))                                  # random index for c

image_test_rand_array = test_image_array[rand]
label_test_rand_array = test_label_array[rand]
prediction_rand_array = np.argmax(model_resnet.predict(image_test_rand_array),axis=1)

plt.figure(figsize=(num_rows*3,num_cols*3))
# fig, axes1 = plt.subplots(num_rows,num_cols,figsize=(num_rows*2,num_cols2))

for i in range(num_rows):
    for j in range(num_cols):
        index = i * num_cols + j
        plt.subplot(num_rows,num_cols,index+1)
        image = image_test_rand_array[index]/255.0  # Extract the image
        label = label_test_rand_array[index]  # Extract the label
        prediction = prediction_rand_array[index]

        # Original pictures (no augmentation layer applied)
        plt.axis("off")
        # Display the image
        plt.imshow(image)
        plt.title(f"Label: {label_dict[label]}\n"
                  f"Predict: {label_dict[prediction]}",
                  fontsize = 8)

plt.tight_layout()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 128ms/step
```

InceptionV3-based model

In [20]:

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.applications.inception_v3 import preprocess_input as inception_preprocess
```

Since the inception model use similar input image shapes as ResNet, I will reuse the processed train, validation, and test sets

Base Model Construction

In [21]:

```
base_model = InceptionV3(weights='imagenet',
                         include_top=False)

# Freeze the weights of the model
```

```
base_model.trainable = False

# # Investigate the structure of the base model and make sure that the weights are frozen
# base_model.summary()
```

Full Model

In [24]:
```
model_inception = models.Sequential([
    layers.Input(TARGET_SIZE + (3,)),
    layers.Lambda(inception_preprocess),
    base_model,
    layers.GlobalAveragePooling2D(),
    layers.BatchNormalization(),
    layers.Dropout(0.5),
    layers.Dense(num_classes, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(num_classes, activation='sigmoid')  # Use 1 for binary classification (dogs vs cats)
])

initial_weight_inception = model_inception.get_weights()

model_inception.summary()
```

Model: "sequential_2"

Layer (type)	**Output Shape**	**Param #**
lambda_2 (Lambda)	(None, 224, 224, 3)	0
inception_v3 (Functional)	(None, 5, 5, 2048)	21,802,784
global_average_pooling2d_2 (GlobalAveragePooling2D)	(None, 2048)	0
batch_normalization_96 (BatchNormalization)	(None, 2048)	8,192
dropout_4 (Dropout)	(None, 2048)	0
dense_4 (Dense)	(None, 37)	75,813
dropout_5 (Dropout)	(None, 37)	0
dense_5 (Dense)	(None, 37)	1,406

Total params: 21,888,195 (83.50 MB)

Trainable params: 81,315 (317.64 KB)

Non-trainable params: 21,806,880 (83.19 MB)

Model Training

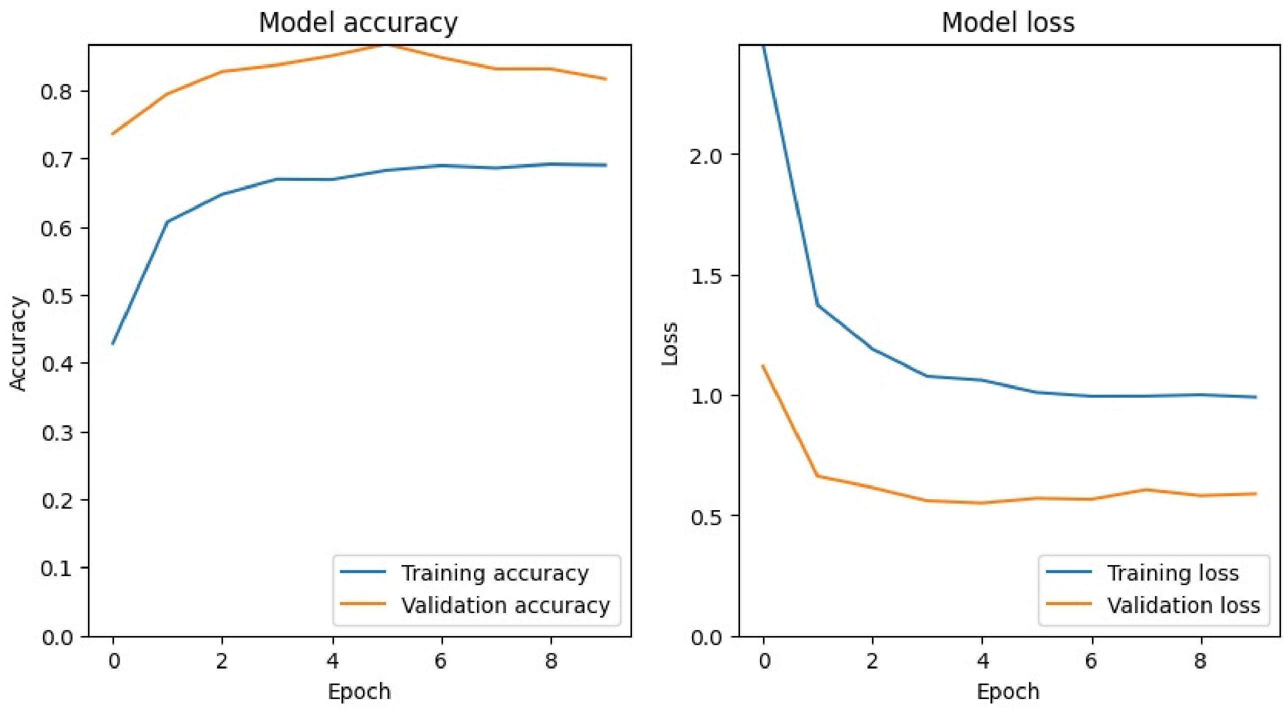
In [25]:
```
model_inception.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.005),
                        loss=tf.keras.losses.SparseCategoricalCrossentropy,
                        metrics=["accuracy"])
loss0, acc0 = model_inception.evaluate(test_image_array,test_label_array)

print("initial loss: {:.2f}".format(loss0))
print("initial accuracy: {:.2f}".format(acc0))
```

```
115/115 ━━━━━━━━━━━━━━━━━━━━ 22s 180ms/step - accuracy: 0.0130 - loss: 3.9160
initial loss: 3.93
initial accuracy: 0.01
```

In [26]:
```
# Train the model with the base layers frozen
initial_epochs = 10
model_inception.set_weights(initial_weight_inception)
history_inception = model_inception.fit(train_image_array,train_label_array,
                                        validation_data=(valid_image_array,valid_label_array),
                                        epochs=initial_epochs)
```

```
Epoch 1/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 25s 236ms/step - accuracy: 0.3012 - loss: 3.2835 - val_accuracy: 0.7364 - val_loss: 1
.1180
Epoch 2/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 22s 236ms/step - accuracy: 0.5874 - loss: 1.4721 - val_accuracy: 0.7948 - val_loss: 0
.6620
Epoch 3/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 22s 236ms/step - accuracy: 0.6440 - loss: 1.2095 - val_accuracy: 0.8274 - val_loss: 0
.6148
Epoch 4/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 22s 240ms/step - accuracy: 0.6618 - loss: 1.0902 - val_accuracy: 0.8370 - val_loss: 0
.5594
Epoch 5/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 22s 234ms/step - accuracy: 0.6772 - loss: 1.0283 - val_accuracy: 0.8505 - val_loss: 0
.5497
Epoch 6/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 21s 233ms/step - accuracy: 0.6888 - loss: 1.0056 - val_accuracy: 0.8682 - val_loss: 0
.5698
Epoch 7/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 21s 232ms/step - accuracy: 0.6882 - loss: 1.0169 - val_accuracy: 0.8478 - val_loss: 0
.5656
Epoch 8/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 21s 233ms/step - accuracy: 0.7027 - loss: 0.9206 - val_accuracy: 0.8315 - val_loss: 0
.6064
Epoch 9/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 21s 232ms/step - accuracy: 0.6878 - loss: 0.9712 - val_accuracy: 0.8315 - val_loss: 0
.5811
Epoch 10/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 22s 235ms/step - accuracy: 0.6820 - loss: 1.0087 - val_accuracy: 0.8166 - val_loss: 0
.5891
```

In [27]:
```python
plot_performance(history_inception)
```




```
The model has a training accuracy of 69.02%
The model has a validation accuracy of 81.66%
```

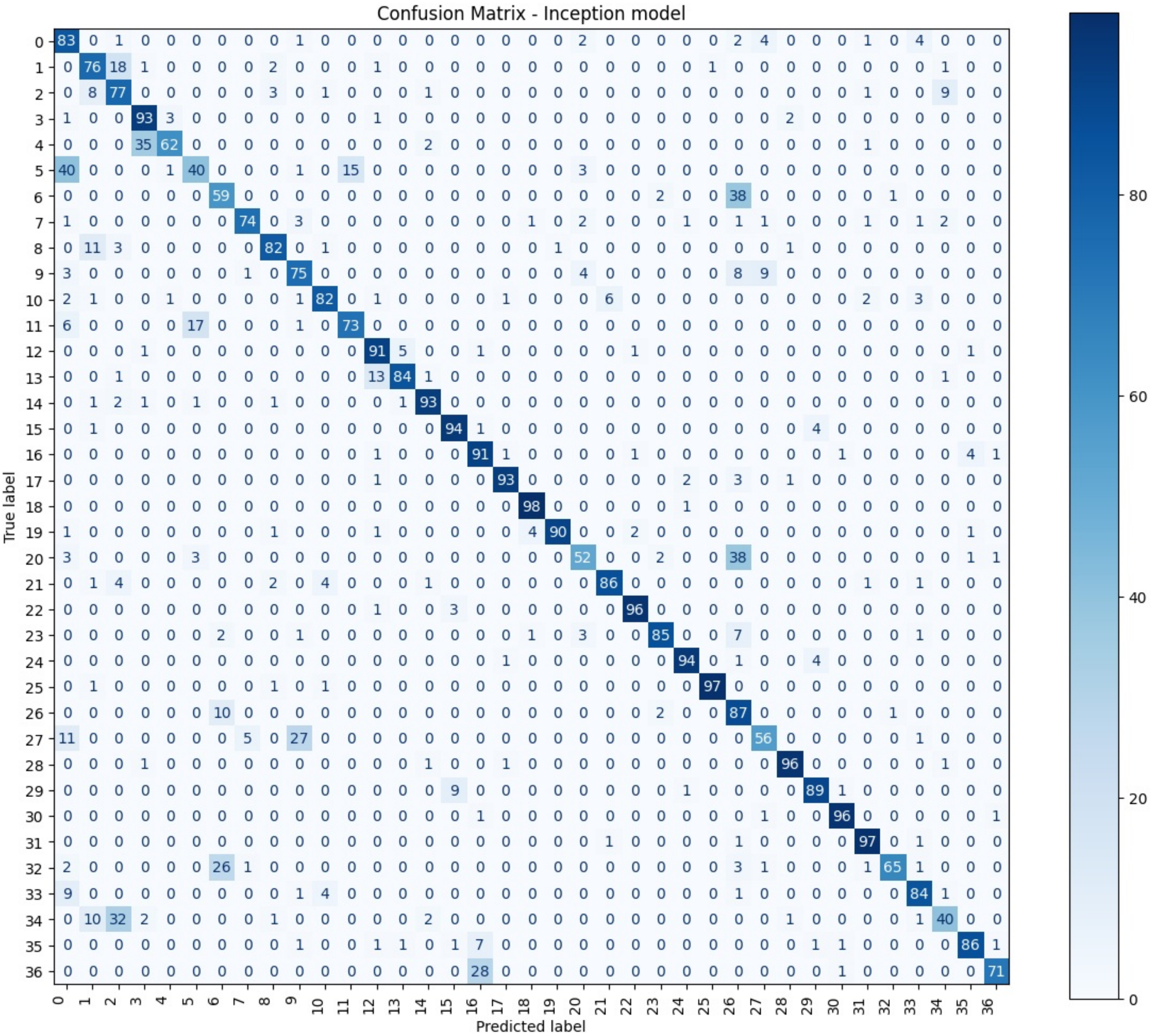
Model Evaluation

In [30]:
```python
test_loss_inception, test_acc_inception = model_inception.evaluate(test_image_array,test_label_array)
print(f"Test accuracy: {test_acc}\n"
      f"Test loss: {test_loss}")

prediction_array_inception = np.argmax(model_inception.predict(test_image_array), axis=1)
```

```
115/115 ━━━━━━━━━━━━━━━━━━━━ 21s 180ms/step - accuracy: 0.8343 - loss: 0.4823
Test accuracy: 0.8135731816291809
Test loss: 0.5614109039306641
115/115 ━━━━━━━━━━━━━━━━━━━━ 23s 189ms/step
```

In [35]:
```python
cm = confusion_matrix(test_label_array, prediction_array_inception)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap=plt.cm.Blues)  # You can change the color map as desired
fig = disp.ax_.get_figure()
fig.set_figwidth(12)
fig.set_figheight(10)
```

```
plt.title("Confusion Matrix - Inception model")
plt.xticks(rotation=90, ha='right')  # Rotate x labels for better readability
plt.yticks(rotation=0)  # Keep y labels horizontal
plt.tight_layout()  # Adjust layout to make room for rotated labels
plt.show()

label_dict
```

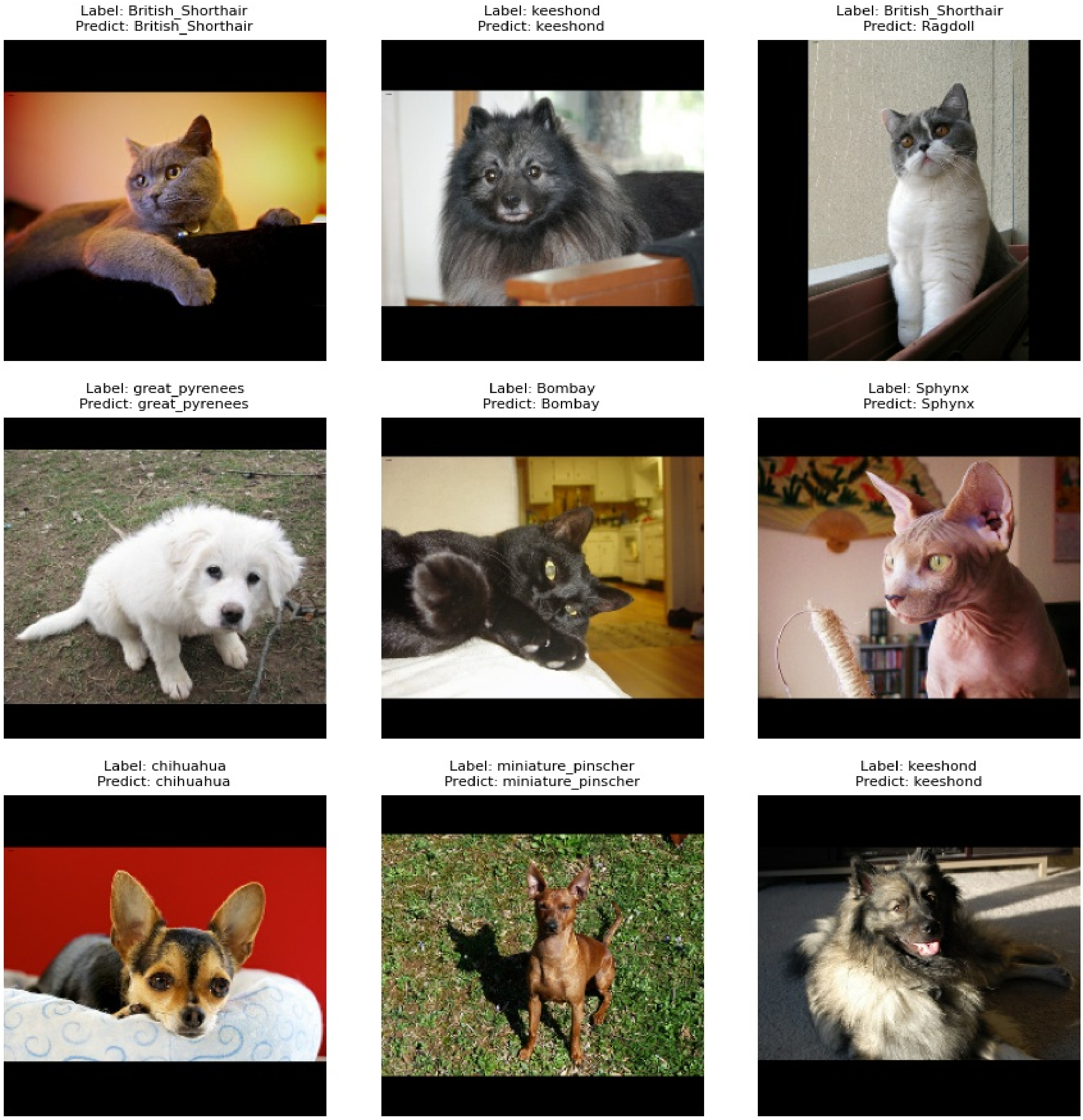


```
Out[35]: {0: 'Abyssinian',
 1: 'american_bulldog',
 2: 'american_pit_bull_terrier',
 3: 'basset_hound',
 4: 'beagle',
 5: 'Bengal',
 6: 'Birman',
 7: 'Bombay',
 8: 'boxer',
 9: 'British_Shorthair',
 10: 'chihuahua',
 11: 'Egyptian_Mau',
 12: 'english_cocker_spaniel',
 13: 'english_setter',
 14: 'german_shorthaired',
 15: 'great_pyrenees',
 16: 'havanese',
 17: 'japanese_chin',
 18: 'keeshond',
 19: 'leonberger',
 20: 'Maine_Coon',
 21: 'miniature_pinscher',
 22: 'newfoundland',
 23: 'Persian',
 24: 'pomeranian',
 25: 'pug',
 26: 'Ragdoll',
 27: 'Russian_Blue',
 28: 'saint_bernard',
 29: 'samoyed',
 30: 'scottish_terrier',
 31: 'shiba_inu',
 32: 'Siamese',
 33: 'Sphynx',
 34: 'staffordshire_bull_terrier',
 35: 'wheaten_terrier',
 36: 'yorkshire_terrier'}
```

Model Prediction and Visualization

```
In [36]: # Sample random images and their indices
num_samples = 9                                                                          # number of samples
num_rows = int(round(sqrt(num_samples))); num_cols = int(num_samples/num_rows)      # number of rows and column:
rand = random.randint(num_test,size = (num_samples))                                   # random index for ci

image_test_rand_array = test_image_array[rand]
label_test_rand_array = test_label_array[rand]
prediction_rand_array = np.argmax(model_inception.predict(image_test_rand_array),axis=1)

plt.figure(figsize=(num_rows*3,num_cols*3))

for i in range(num_rows):
    for j in range(num_cols):
        index = i * num_cols + j
        plt.subplot(num_rows,num_cols,index+1)
        image = image_test_rand_array[index]/255.0  # Extract the image
        label = label_test_rand_array[index]  # Extract the label
        prediction = prediction_rand_array[index]

        # Original pictures (no augmentation layer applied)
        plt.axis("off")
        # Display the image
        plt.imshow(image)
        plt.title(f"Label: {label_dict[label]}\n"
                  f"Predict: {label_dict[prediction]}",
                  fontsize = 8)

plt.tight_layout()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 87ms/step
```



ResNet vs Inception Comparison

Model Performance Comparison

In [50]:
```
test_loss_resnet, test_acc_resnet = model_resnet.evaluate(test_image_array,test_label_array)
test_loss_inception, test_acc_inception = model_inception.evaluate(test_image_array,test_label_array)
```

```
115/115 ━━━━━━━━━━━━━━━━━━━━ 40s 346ms/step - accuracy: 0.8325 - loss: 0.4967
115/115 ━━━━━━━━━━━━━━━━━━━━ 22s 191ms/step - accuracy: 0.8343 - loss: 0.4823
```

In [56]:
```
msg_loss = "comparable"; msg_acc = "comparable"
test_loss_diff = test_loss_resnet - test_loss_inception
test_acc_diff = test_acc_resnet - test_acc_inception
diff_threshold = 0.01

if np.abs(test_loss_diff) > diff_threshold:
```

```python
    if test_loss_resnet > test_loss_inception: msg_loss = "better"
    else: msg_loss = "worse"

if np.abs(test_acc_diff) > diff_threshold:
    if test_acc_resnet > test_acc_inception: msg_acc = "better"
    else: msg_acc = "worse"

print(f"The ResNet-based model has {msg_acc} accuracy compared to the Inception-based model\n"
      f"Resnet-based model accuracy: {test_acc_resnet*100:.2f}%\n"
      f"Inception-based model accuracy: {test_acc_inception*100:.2f}%")

print(f"The ResNet-based model has {msg_loss} accuracy compared to the Inception-based model\n"
      f"Resnet-based model loss: {test_loss_resnet*100:.2f}%\n"
      f"Inception-based model loss: {test_loss_inception*100:.2f}%")
```

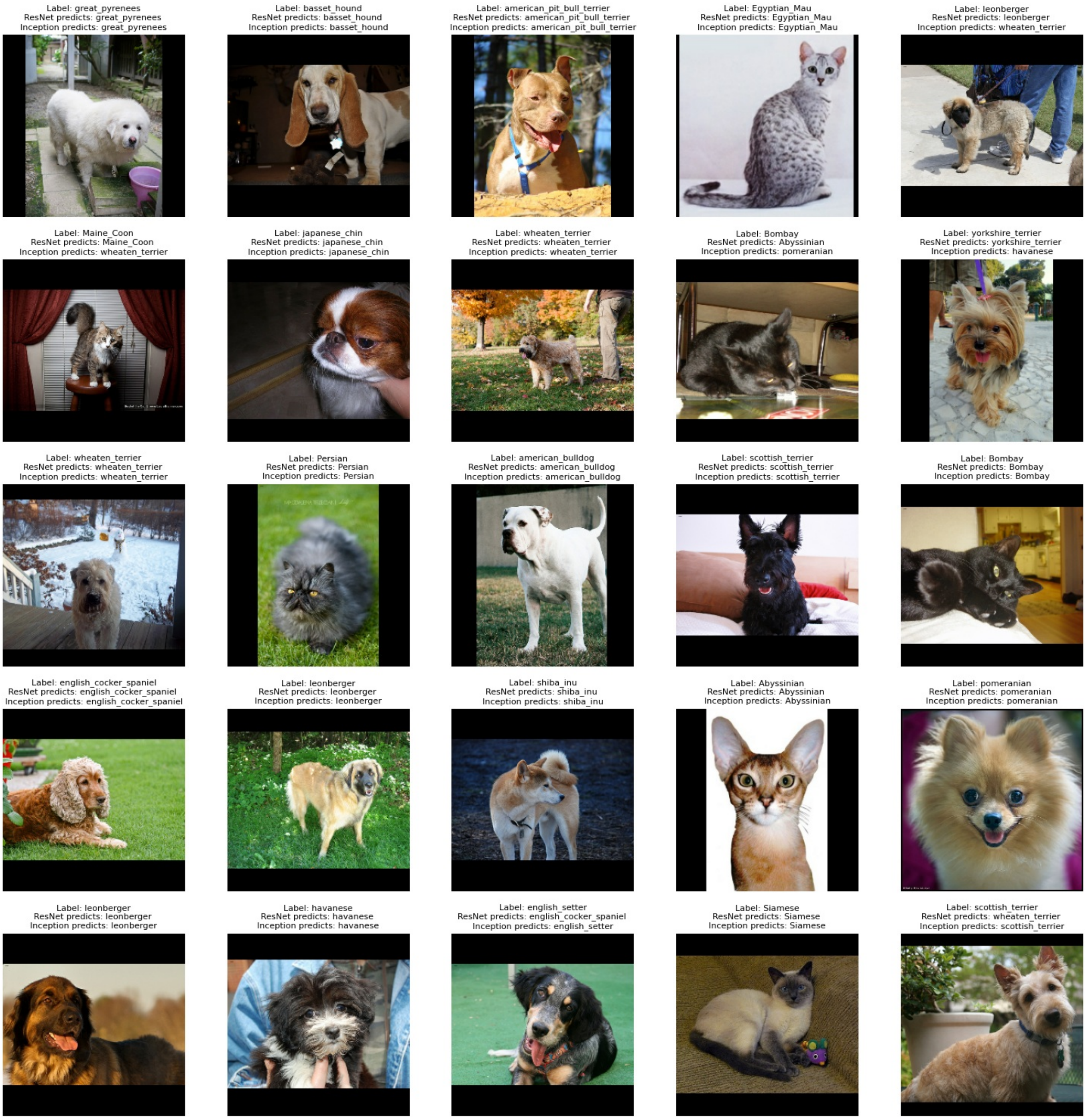
```
The ResNet-based model has comparable accuracy compared to the Inception-based model
Resnet-based model accuracy: 83.32%
Inception-based model accuracy: 83.29%
The ResNet-based model has comparable accuracy compared to the Inception-based model
Resnet-based model loss: 50.15%
Inception-based model loss: 50.80%
```

Prediction Comparison

In [57]:

```python
# Sample random images and their indices
num_samples = 25                                                                         # number of sample
num_rows = int(round(sqrt(num_samples))); num_cols = int(num_samples/num_rows)      # number of rows and column
rand = random.randint(num_test,size = (num_samples))                                   # random index for ci

image_test_rand_array = test_image_array[rand]
label_test_rand_array = test_label_array[rand]
prediction_rand_array_resnet = np.argmax(model_resnet.predict(image_test_rand_array),axis=1)
prediction_rand_array_inception = np.argmax(model_inception.predict(image_test_rand_array),axis=1)

plt.figure(figsize=(num_rows*3,num_cols*3))
# fig, axes1 = plt.subplots(num_rows,num_cols,figsize=(num_rows*2,num_cols2))

for i in range(num_rows):
    for j in range(num_cols):
        index = i * num_cols + j
        plt.subplot(num_rows,num_cols,index+1)
        image = image_test_rand_array[index]/255.0      # Extract the image
        label = label_test_rand_array[index]            # Extract the label
        prediction_resnet = prediction_rand_array_resnet[index]
        prediction_inception = prediction_rand_array_inception[index]

        # Original pictures (no augmentation layer applied)
        plt.axis("off")
        # Display the image
        plt.imshow(image)
        plt.title(f"Label: {label_dict[label]}\n"
                  f"ResNet predicts: {label_dict[prediction_resnet]} \n"
                  f"Inception predicts: {label_dict[prediction_inception]}",
                  fontsize = 8)

plt.tight_layout()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 287ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 160ms/step
```


Label: great_pyrenees
ResNet predicts: great_pyrenees
Inception predicts: great_pyrenees
Label: basset_hound
ResNet predicts: basset_hound
Inception predicts: basset_hound
Label: american_pit_bull_terrier
ResNet predicts: american_pit_bull_terrier
Inception predicts: american_pit_bull_terrier
Label: Egyptian_Mau
ResNet predicts: Egyptian_Mau
Inception predicts: Egyptian_Mau
Label: leonberger
ResNet predicts: leonberger
Inception predicts: wheaten_terrier
Label: Maine_Coon
ResNet predicts: Maine_Coon
Inception predicts: wheaten_terrier
Label: japanese_chin
ResNet predicts: japanese_chin
Inception predicts: japanese_chin
Label: wheaten_terrier
ResNet predicts: wheaten_terrier
Inception predicts: wheaten_terrier
Label: Bombay
ResNet predicts: Abyssinian
Inception predicts: pomeranian
Label: yorkshire_terrier
ResNet predicts: yorkshire_terrier
Inception predicts: havanese
Label: wheaten_terrier
ResNet predicts: wheaten_terrier
Inception predicts: wheaten_terrier
Label: Persian
ResNet predicts: Persian
Inception predicts: Persian
Label: american_bulldog
ResNet predicts: american_bulldog
Inception predicts: american_bulldog
Label: scottish_terrier
ResNet predicts: scottish_terrier
Inception predicts: scottish_terrier
Label: Bombay
ResNet predicts: Bombay
Inception predicts: Bombay
Label: english_cocker_spaniel
ResNet predicts: english_cocker_spaniel
Inception predicts: english_cocker_spaniel
Label: leonberger
ResNet predicts: leonberger
Inception predicts: leonberger
Label: shiba_inu
ResNet predicts: shiba_inu
Inception predicts: shiba_inu
Label: Abyssinian
ResNet predicts: Abyssinian
Inception predicts: Abyssinian
Label: pomeranian
ResNet predicts: pomeranian
Inception predicts: pomeranian
Label: leonberger
ResNet predicts: leonberger
Inception predicts: leonberger
Label: havanese
ResNet predicts: havanese
Inception predicts: havanese
Label: english_setter
ResNet predicts: english_cocker_spaniel
Inception predicts: english_setter
Label: Siamese
ResNet predicts: Siamese
Inception predicts: Siamese
Label: scottish_terrier
ResNet predicts: wheaten_terrier
Inception predicts: scottish_terrier