Project 1: Oxford-IIIT Pet Dataset Classification Challenge

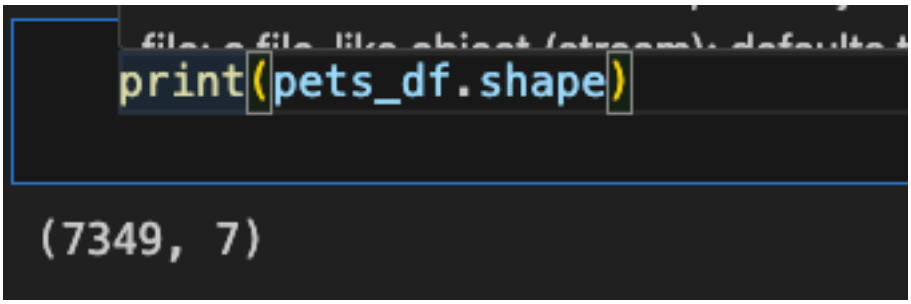
Image classification

Kian Sahafi
Dr.Mohsen Afsharchi
November 30, 2023

1.Data preparation:

the dataset consists of 37 classes (or types) of dog/cat breed. And the distribution consists of 200 pictures for each class. I have used the method of augmentation to improve my models robustness. As can be seen in this picture:

	full_path	file	file_name	label_name	label_id	species	breed_id
0	/kaggle/working/images/wheaten_terrier_56.jpg	wheaten_terrier_56.jpg	wheaten_terrier_56	wheaten_terrier	36	2	24
1	/kaggle/working/images/Sphynx_105.jpg	Sphynx_105.jpg	Sphynx_105	Sphynx	34	1	12
2	/kaggle/working/images/great_pyrenees_200.jpg	great_pyrenees_200.jpg	great_pyrenees_200	great_pyrenees	16	2	10
3	/kaggle/working/images/Bengal_179.jpg	Bengal_179.jpg	Bengal_179	Bengal	6	1	2
4	/kaggle/working/images/shiba_inu_141.jpg	shiba_inu_141.jpg	shiba_inu_141	shiba_inu	32	2	22
5	/kaggle/working/images/wheaten_terrier_163.jpg	wheaten_terrier_163.jpg	wheaten_terrier_163	wheaten_terrier	36	2	24
6	/kaggle/working/images/Russian_Blue_89.jpg	Russian_Blue_89.jpg	Russian_Blue_89	Russian_Blue	28	1	10
7	/kaggle/working/images/pomeranian_159.jpg	pomeranian_159.jpg	pomeranian_159	pomeranian	25	2	17
8	/kaggle/working/images/Bombay_87.jpg	Bombay_87.jpg	Bombay_87	Bombay	8	1	4
9	/kaggle/working/images/Sphynx_175.jpg	Sphynx_175.jpg	Sphynx_175	Sphynx	34	1	12



It can be seen that the we have 7349 data to work with

The issue I had with the augmentation was that I was augmenting the actual data and I was not helping me improve the models performance as I wanted.

So after researching a ton, I found out that I should make a copy of the original data and do the augmentation on those and add them to the original data. With this approach not only I had improved the performance, but also I had generated more data to train my model with.

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing import image

# Assuming pets_df is your original dataframe
# Split the dataframe into training and validation sets
train_df, valid_df = train_test_split(pets_df, test_size=0.2, random_state=42)

# Create a copy of the training dataframe for augmentation
augmented_train_df = train_df.copy()

datagen = image.ImageDataGenerator(rescale=1/255.,
                                   validation_split=0.2,
                                   rotation_range=20,
                                   width_shift_range=0.2,
                                   height_shift_range=0.2,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   horizontal_flip=True,
                                   fill_mode='nearest')

# Create an ImageDataGenerator for augmentation
augmentation_datagen = image.ImageDataGenerator(
    rescale=1/255.,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

# Generate augmented images
augmented_train_generator = augmentation_datagen.flow_from_dataframe(
    dataframe=augmented_train_df,
    directory='/kaggle/working/images',
    x_col='file',
    y_col='label_name',
    target_size=(224, 224),
    class_mode='categorical',
    batch_size=256,
    seed=42  # Set seed for reproducibility
)
```

(variable) augmentation_datagen: Any

```python
# Concatenate the original and augmented dataframes for training
final_train_df = pd.concat([train_df, augmented_train_df])

# Create generators for training and validation using the concatenated dataframes
final_train_generator = datagen.flow_from_dataframe(
    dataframe=final_train_df,
    directory='/kaggle/working/images',
    x_col='file',
    y_col='label_name',
    target_size=(224, 224),
    class_mode='categorical',
    batch_size=256,
    seed=42  # Set seed for reproducibility
)

final_validation_generator = datagen.flow_from_dataframe(
    dataframe=valid_df,
    directory='/kaggle/working/images',
    x_col='file',
    y_col='label_name',
    target_size=(224, 224),
    class_mode='categorical',
    batch_size=256,
    seed=42
)
```

```
Found 5879 validated image filenames belonging to 37 classes.
Found 11758 validated image filenames belonging to 37 classes.
Found 1470 validated image filenames belonging to 37 classes.
```

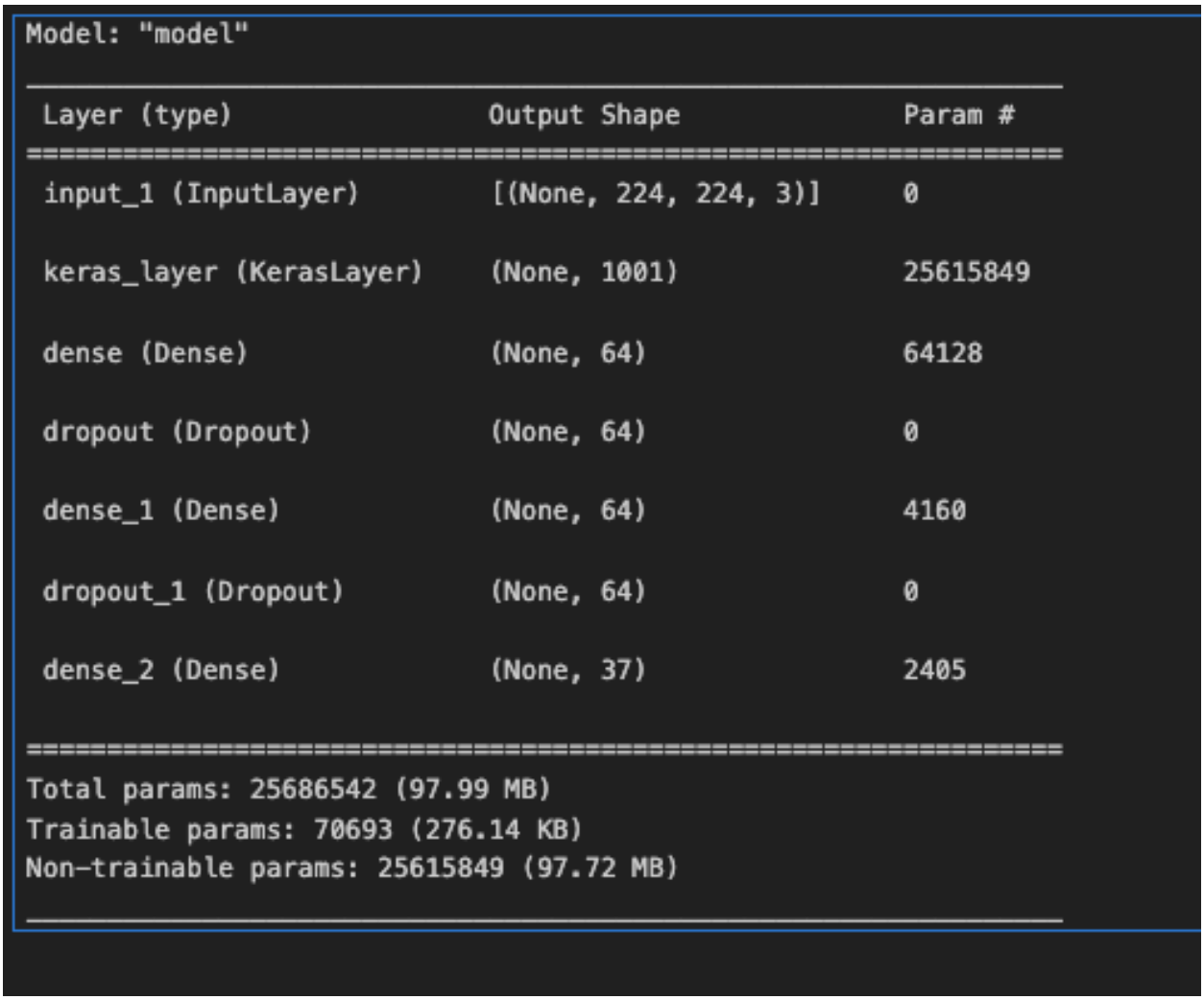
You can clearly see that I have increased the training data to 11,758 training datum. And 1,478 validation datum.

2. Model Development

for the model I have tried to create the architecture of the model myself but the accuracy was far away from what I wanted to achieve. So I tried using some pre-trained models such as VGG-16 and VGG-19 which they did not give the accuracy I was hoping for. So I stumbled upon the mighty ResNet50-V2 which lead to some quite amazing results. And I started to develop my models architecture with this pre-trained model.

```python
from keras.layers import Dropout, Flatten, Dense, Conv2D, MaxPooling2D
from keras.models import Sequential
import keras
from keras import optimizers, models, layers

def create_model(trainable):
    resnet50 = 'https://tfhub.dev/google/imagenet/resnet_v2_50/classification/5'
    ResNet = hub.KerasLayer(resnet50, trainable = trainable)
    tf.keras.backend.clear_session()

    data = tf.keras.Input(shape=(224,224,3)) # input shape
    L1 = ResNet(data) # ResNet layer


    # relu bood ke 89% acc dad
    # sigmoid zadam acc shod 79%

    L2 = Dense(64, activation='relu')(L1) # Fully conected layer
    L2 = Dropout(0.3)(L2) # for reducing overfitting

    L3 = Dense(64, activation='relu')(L2) # Fully conected layer
    L3 = Dropout(0.3)(L3)

    out = Dense(37, activation='softmax')(L3) # Output layer for classification

    model = tf.keras.Model(data, out) # build model

    lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
        initial_learning_rate=1e-2,
#         decay_steps=10,
        decay_steps=10000,
        decay_rate=0.9)

    #TODO: l2 regularization

    # optimizer = tf.keras.optimizers.SGD(learning_rate=lr_schedule)
    optimizer = tf.keras.optimizers.Adam(learning_rate=lr_schedule)
    model.compile(loss = 'categorical_crossentropy', optimizer = 'adam', metrics = ['accuracy']) # config model
    return model

model = create_model(False)
model.summary()
```

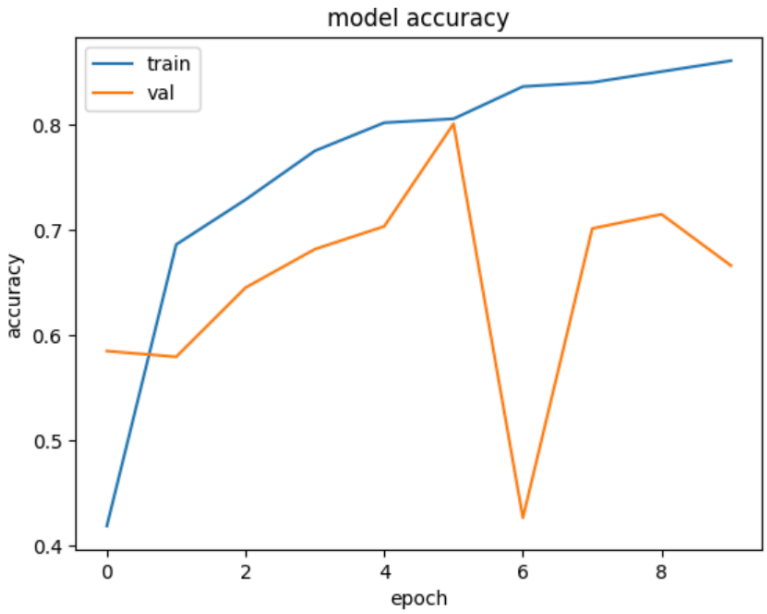
You can see that I put the ResNet50-V2 in the front of my neural network, followed by two other fully connected layers with “Relu" activation function (the reason for choosing this was completely experimental) with dropout after each one (for reducing overfitting) and at last an output layer with 37 cells for our 37 classes. At last I created a learning rate scheduler which slow downs the learning after a certain amount of training. The optimizer that I used was also experimental. First I used the SGD optimizer (gradient descent) which was not ideal because it kept getting stuck on 82% accuracy. But after switching to ADAM optimizer I could go up to 87%.

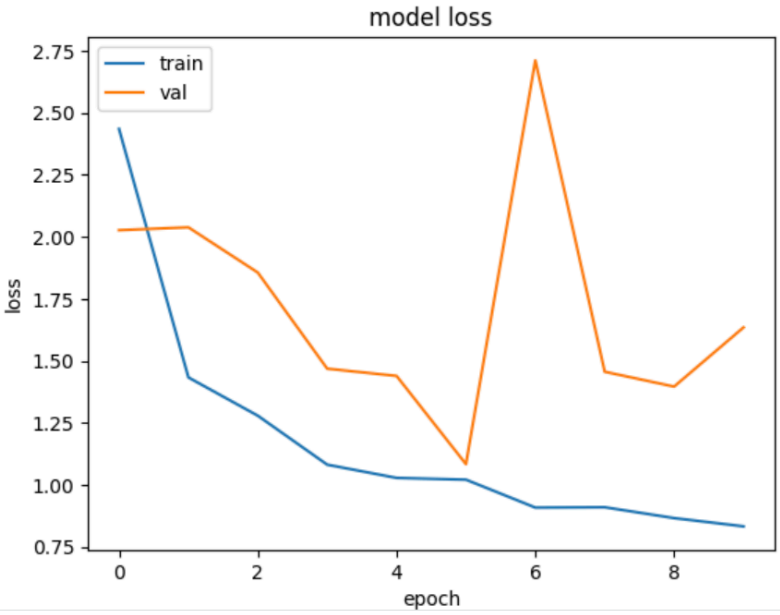


```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 keras_layer (KerasLayer)    (None, 1001)              25615849

 dense (Dense)               (None, 64)                64128

 dropout (Dropout)           (None, 64)                0

 dense_1 (Dense)             (None, 64)                4160

 dropout_1 (Dropout)         (None, 64)                0

 dense_2 (Dense)             (None, 37)                2405

=================================================================
Total params: 25686542 (97.99 MB)
Trainable params: 70693 (276.14 KB)
Non-trainable params: 25615849 (97.72 MB)
_________________________________________________________________
```

As can be seen in the picture I have made the ResNet50-V2 untrainable because the computer that I was doing to training with was not capable of training such large quantities of parameters. So I figured I would freeze the ResNet50-V2 at first and do the first training on them and then save the weights of my trained neural network and the load them again for the second time and unfreeze the ResNet50-V2. But you can see in the future that I was not quite successful in this action.

The reason for choosing the learning rate was also experimental. You can see the result of setting the initial learning rate (1e-1):





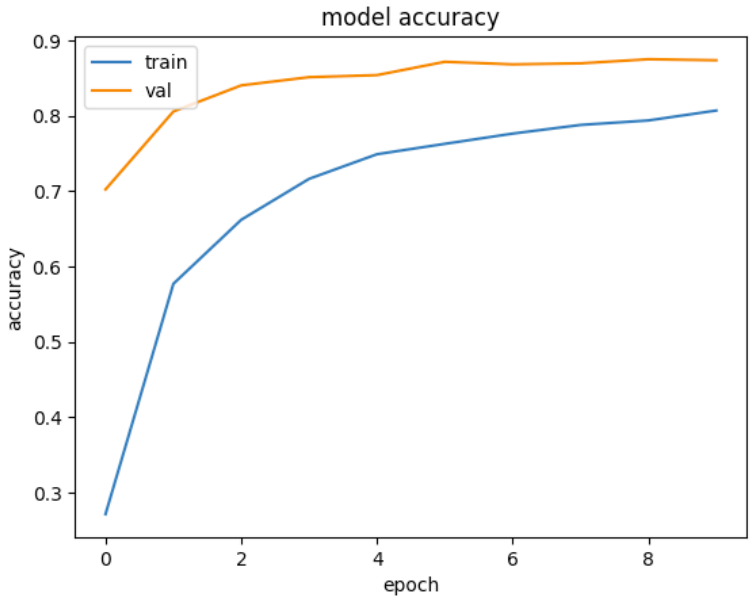
3. Training and Evaluation

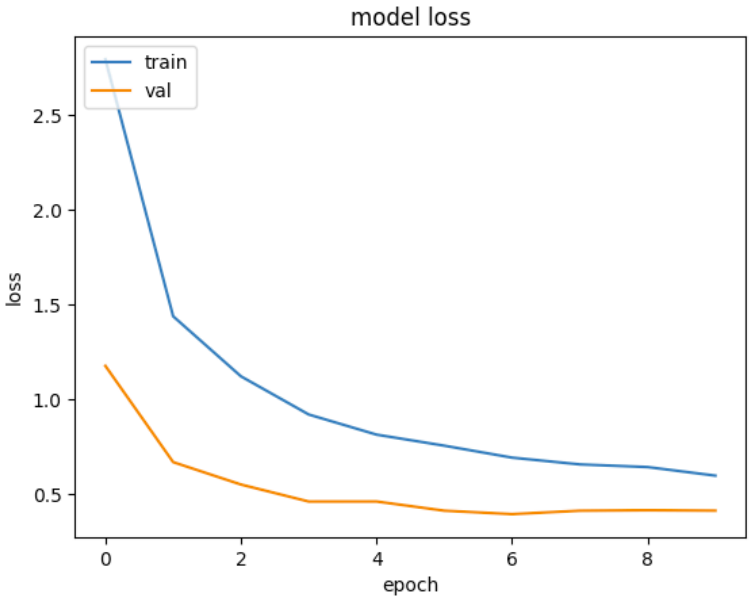
```
history = model.fit(
    final_train_generator,
    validation_data = final_validation_generator,
    epochs=10,
    callbacks=[tensorboard_callback]
)
```

```
Epoch 1/10
46/46 [==============================] - 226s 5s/step - loss: 2.7928 - accuracy: 0.2711 - val_loss: 1.1767 - val_accuracy: 0.7020
Epoch 2/10
46/46 [==============================] - 211s 5s/step - loss: 1.4380 - accuracy: 0.5768 - val_loss: 0.6694 - val_accuracy: 0.8054
Epoch 3/10
46/46 [==============================] - 211s 5s/step - loss: 1.1216 - accuracy: 0.6618 - val_loss: 0.5512 - val_accuracy: 0.8401
Epoch 4/10
46/46 [==============================] - 212s 5s/step - loss: 0.9204 - accuracy: 0.7161 - val_loss: 0.4616 - val_accuracy: 0.8510
Epoch 5/10
46/46 [==============================] - 211s 5s/step - loss: 0.8143 - accuracy: 0.7488 - val_loss: 0.4619 - val_accuracy: 0.8537
Epoch 6/10
46/46 [==============================] - 212s 5s/step - loss: 0.7560 - accuracy: 0.7625 - val_loss: 0.4134 - val_accuracy: 0.8714
Epoch 7/10
46/46 [==============================] - 215s 5s/step - loss: 0.6930 - accuracy: 0.7762 - val_loss: 0.3953 - val_accuracy: 0.8680
Epoch 8/10
46/46 [==============================] - 214s 5s/step - loss: 0.6574 - accuracy: 0.7876 - val_loss: 0.4133 - val_accuracy: 0.8694
Epoch 9/10
46/46 [==============================] - 213s 5s/step - loss: 0.6433 - accuracy: 0.7936 - val_loss: 0.4156 - val_accuracy: 0.8748
Epoch 10/10
46/46 [==============================] - 215s 5s/step - loss: 0.5981 - accuracy: 0.8068 - val_loss: 0.4136 - val_accuracy: 0.8735
```

It can be seen from the figure that I have achieved the validation accuracy of 87% in the 10th Epoch.

And these are the accuracy and loss results:





After that I attempted to make the same architecture and load the previously trained weights into it and the train it again with the ResNet50-V2 not frozen this time. But whatever I tried to I could not load the weights into the newly created model.

This is the attempt I was trying to do to load the previously trained weights:

Load the previously saved weights

```
latest = tf.train.latest_checkpoint(checkpoint_dir)
model.load_weights(latest)
```

[28]

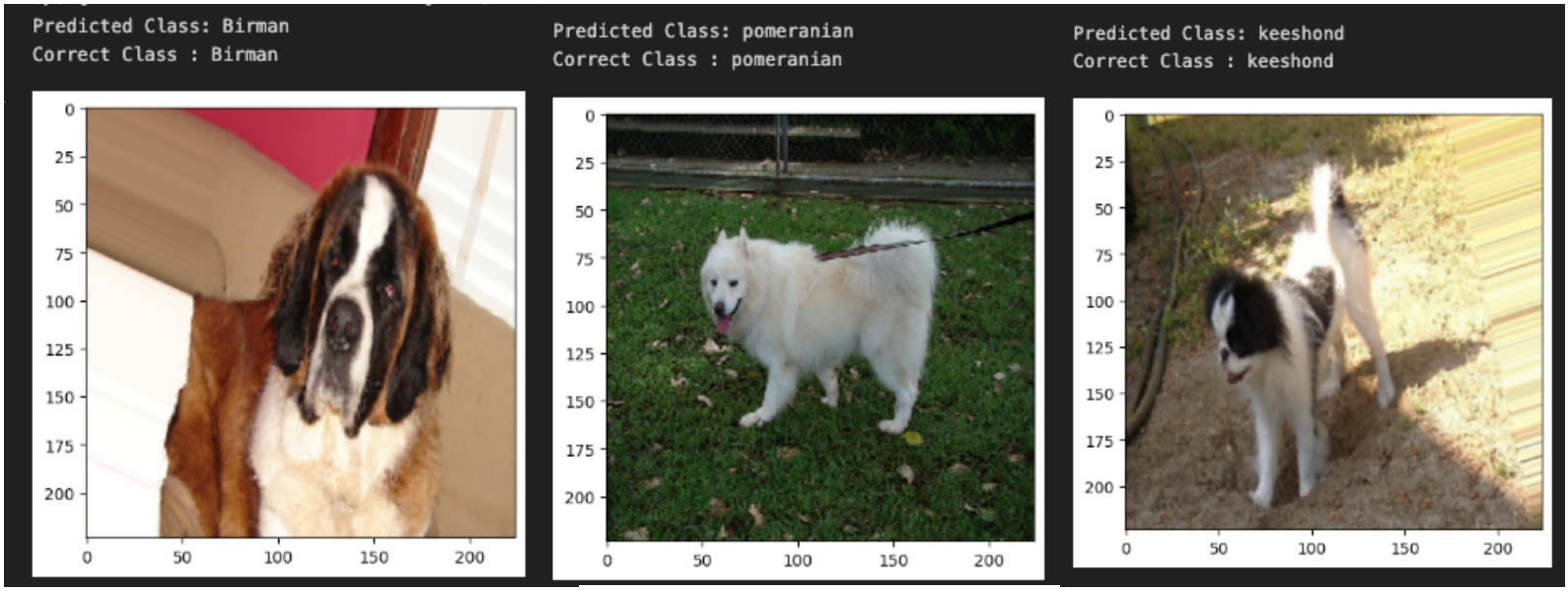
```
<tensorflow.python.checkpoint.checkpoint.CheckpointLoadStatus at 0x7bca18382950>
```

```
# loss, acc = model.evaluate(train_generator, pets_df['label'], verbose=2)
# print("Restored model, accuracy: {:5.2f}%".format(100 * acc))
```

[29]

```
history = model.fit(
    final_train_generator,
    validation_data = final_validation_generator,
    epochs=10,
    callbacks=[tensorboard_callback]
)
```

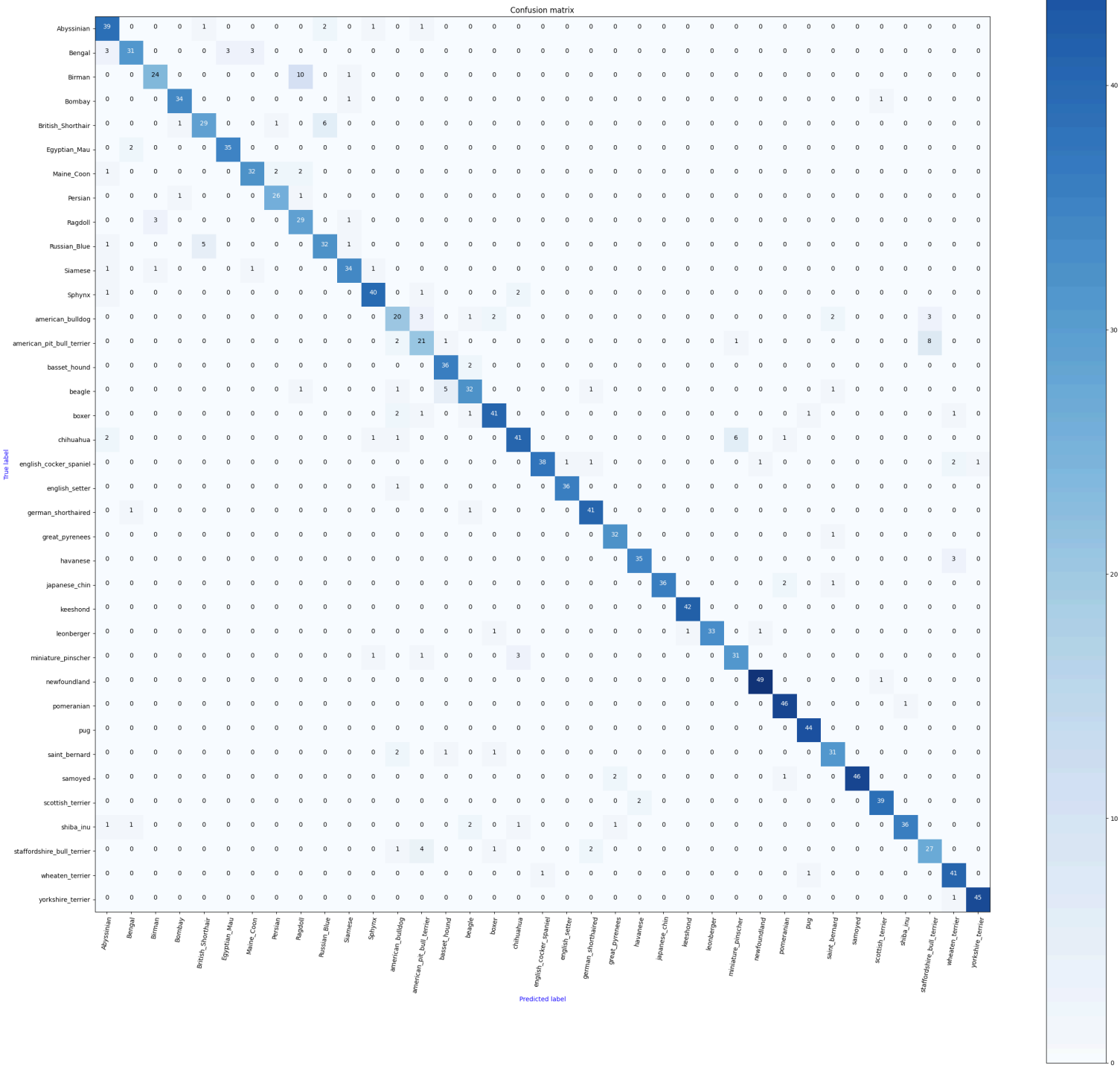
Sample outputs:



Let’s see some Precision, Recall, F1-Score:

	precision	recall	f1-score	support
Abyssinian	0.80	0.89	0.84	44
Bengal	0.89	0.78	0.83	40
Birman	0.86	0.69	0.76	35
Bombay	0.94	0.94	0.94	36
British_Shorthair	0.83	0.78	0.81	37
Egyptian_Mau	0.92	0.95	0.93	37
Maine_Coon	0.89	0.86	0.88	37
Persian	0.90	0.93	0.91	28
Ragdoll	0.67	0.88	0.76	33
Russian_Blue	0.80	0.82	0.81	39
Siamese	0.89	0.89	0.89	38
Sphynx	0.91	0.91	0.91	44
american_bulldog	0.67	0.65	0.66	31
american_pit_bull_terrier	0.66	0.64	0.65	33
basset_hound	0.84	0.95	0.89	38
beagle	0.82	0.78	0.80	41
boxer	0.89	0.87	0.88	47
chihuahua	0.87	0.79	0.83	52
english_cocker_spaniel	0.97	0.86	0.92	44
english_setter	0.97	0.97	0.97	37
german_shorthaired	0.91	0.95	0.93	43
great_pyrenees	0.91	0.97	0.94	33
...				
accuracy			0.89	1470
macro avg	0.88	0.88	0.88	1470
weighted avg	0.89	0.89	0.89	1470

And last but not least, confusion matrix:



It can be seen in the matrix that the neural network is doing a somewhat good job on finding the right breed of the animals. But it confuses Ragdalls with Birmans & American pit bull terrier with Staffordshire terrier and for solving this issue we could increase the distribution for these two breeds but obviously this was not an option because we cannot change the dataset.