CSE 598 Intro to Deep Learning

Name: Chia-Cheng Kuo, ID: 1222458406

Project Proposal

Topic: Recognizing Dog Breed Using Deep Learning

I. General project idea:

Using computer vision to recognize dog breeds.

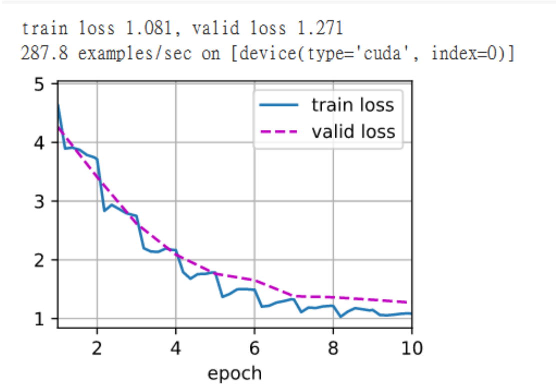
II. The problem:

A. The Dog Breed Identification Competition on Kaggle:
   https://www.kaggle.com/c/dog-breed-identification/overview/description

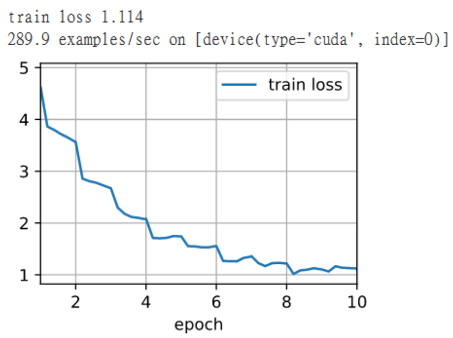
B. Link of an example provided in the professor's slide:
   https://d2l.ai/chapter_computer-vision/kaggle-dog.html

Result of implementing the example:

Training and Validating the Model



Classifying the Testing Set

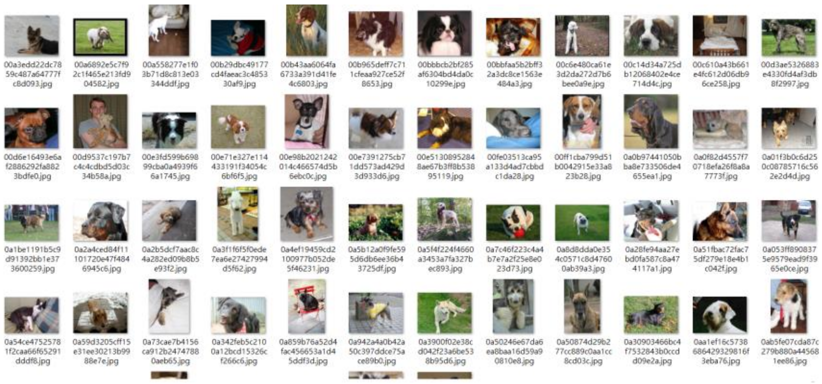


III. Dataset:

The dataset is provided on Kaggle competition website:
https://www.kaggle.com/c/dog-breed-identification/data

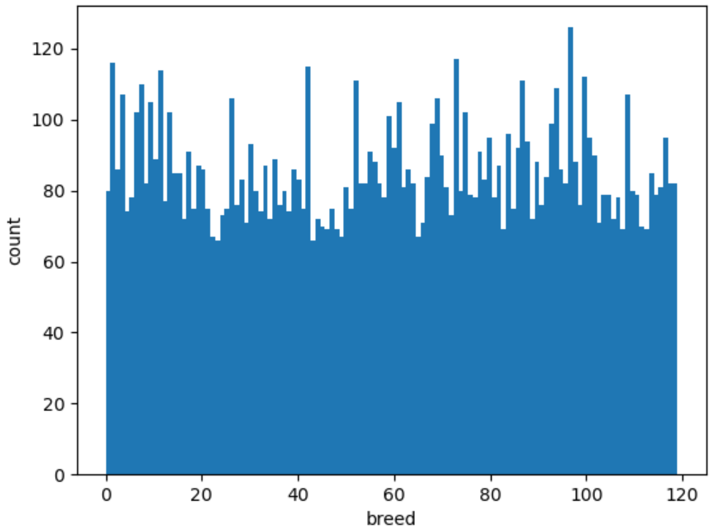
Training dataset (10.2k files):

Testing dataset (10.4k files):



Labels: 120 breeds

Distribution of breeds: (using matplotlib.pyplot -> plt.hist(train_y, 120))



IV. Baselines and Evaluation Metrics

I used DummyClassifier to do the baseline, using “most_frequent”.

Then use classification_report(...) from sklearn to evaluate the baseline.

```
X = y_test
y = y_test
dummy_clf = DummyClassifier(strategy="most_frequent")
dummy_clf.fit(X, y)
print(f"breed which dummyClassifier predict: {dummy_clf.predict(X)[0]} -> {breed_dict[dummy_clf.predict(X)[0]]}")
#print(dummy_clf.score(X, y))
x_dummy=dummy_clf.predict(X)
print(classification_report(y_test, x_dummy, target_names = classes))
```

```
Total number of unique Dog Breeds : 120
length of label classes: 60
{0: 'afghan_hound', 1: 'airedale', 2: 'appenzeller', 3: 'basenji', 4: 'beagle', 5: 'bernese_mountain_dog', 6: 'blenheim_spaniel', 7: 'bluetick', 8: 'border_terrier', 9: 'boston_bull', 10: 'boxer', 11: 'briard', 12: 'bull_mastiff', 13: 'cardigan', 14: 'chihuahua', 15: 'clumber', 16: 'collie', 17: 'dandie_dinmont', 18: 'dingo', 19: 'english_foxhound', 20: 'english_springer', 21: 'eskimo_dog', 22: 'french_bulldog', 23: 'german_short-haired_pointer', 24: 'golden_retriever', 25: 'great_dane', 26: 'greater_swiss_mountain_dog', 27: 'ibizan_hound', 28: 'irish_terrier', 29: 'irish_wolfhound', 30: 'japanese_spaniel', 31: 'kelpie', 32: 'komondor', 33: 'labrador_retriever', 34: 'leonberg', 35: 'malamute', 36: 'maltese_dog', 37: 'miniature_pinscher', 38: 'miniature_schnauzer', 39: 'norfolk_terrier', 40: 'norwich_terrier', 41: 'otterhound', 42: 'pekinese', 43: 'pomeranian', 44: 'redbone', 45: 'rottweiler', 46: 'saluki', 47: 'schipperke', 48: 'scottish_deerhound', 49: 'shetland_sheepdog', 50: 'siberian_husky', 51: 'soft-coated_wheaten_terrier', 52: 'standard_poodle', 53: 'sussex_spaniel', 54: 'tibetan_terrier', 55: 'toy_terrier', 56: 'walker_hound', 57: 'welsh_springer_spaniel', 58: 'whippet', 59: 'yorkshire_terrier'}
breed which dummyClassifier predict: 43 -> pomeranian
                          precision    recall  f1-score   support

            afghan_hound       0.00      0.00      0.00        27
                airedale       0.00      0.00      0.00        19
```

otterhound	0.00	0.00	0.00	12
pekinese	0.00	0.00	0.00	16
pomeranian	0.03	1.00	0.05	29
redbone	0.00	0.00	0.00	11
rottweiler	0.00	0.00	0.00	14

The dummy classifier only got some labels which are Pomeranian correct (other predictions are wrong), because it predicts all labels as Pomeranian.

Accuracy of Dummy Classifier (using most frequent):

	precision	recall	f1-score	support
accuracy			0.03	1035
macro avg	0.00	0.02	0.00	1035
weighted avg	0.00	0.03	0.00	1035

Accuracy of Dummy Classifier (using stratified):

	precision	recall	f1-score	support
accuracy			0.02	1035
macro avg	0.02	0.02	0.02	1035
weighted avg	0.02	0.02	0.02	1035:

V. Experiments and Results

I tried using the example model from this website:
https://techvidvan.com/tutorials/dog-breed-classification/

The example model of the referenced website can only train the model, then predict one picture of a dog.

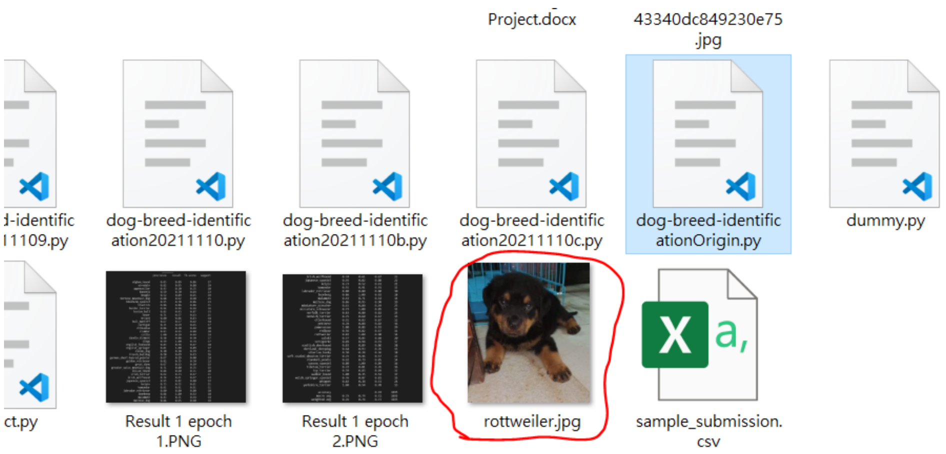
Below is the testing part of the original code:

```
#Save the model for prediction
model.save("model")

#load the model
model = load_model("model")

#get the image of the dog for prediction
pred_img_path = 'rottweiler.jpg'
#read the image file and convert into numeric format
#resize all images to one dimension i.e. 224x224
pred_img_array = cv2.resize(cv2.imread(pred_img_path,cv2.IMREAD_COLOR),((im_size,im_size)))
#scale array into the range of -1 to 1.
#expand the dimesion on the axis 0 and normalize the array values
pred_img_array = preprocess_input(np.expand_dims(np.array(pred_img_array[...,::-1].astype(np.float32)).copy(), axis=0))

#feed the model with the image array for prediction
pred_val = model.predict(np.array(pred_img_array,dtype="float32"))
#display the image of dog
cv2.imshow("TechVidvan",cv2.resize(cv2.imread(pred_img_path,cv2.IMREAD_COLOR),((im_size,im_size))))
#display the predicted breed of dog
pred_breed = sorted(new_list)[np.argmax(pred_val)]
print("Predicted Breed for this Dog is :",pred_breed)
```



↑ The original model from the referenced website can only predict one picture of a dog.

I adjusted how the program tests the model.

I stored the predicted labels into a list, then use classification_report. (Because the testing data set given by the website has no label, I split testing data from training data)

```
all_preds = []
for test_img in x_test:

    #feed the model with the image array for prediction
    test_img = test_img.reshape(1,224,224,3)
    pred_val = model.predict(np.array(test_img,dtype="float32"))

    #display the predicted breed of dog
    pred_breed = sorted(new_list)[np.argmax(pred_val)]
    #print("Predicted Breed for this Dog is :",pred_breed)
    #print(f"breed number: {np.argmax(pred_val)}")
    all_preds .append(np.argmax(pred_val))
#print (all_preds)
```

Result:

A. Using the codes from the website

I tested the model using different hyperparameters, including: train-test split ratio, batch size, epochs, learning rates, image augmentation.

The original example code uses below image augmentation:

train_datagen = ImageDataGenerator(rotation_range=45, width_shift_range=0.2, height_shift_range=0.2, shear_range=0.2, zoom_range=0.25, horizontal_flip=True, fill_mode='nearest')

Using CPU: 11th Gen Intel® Core™ i5-1135G7 @2.4GHz

length(number) of train set data: 8177

length(number) of test set data: 2045

Epoch= 25, Learning rate= 0.001

	precision	recall	f1-score	support
accuracy			0.74	2045

	precision	recall	f1-score	support
macro avg	0.74	0.74	0.73	2045
weighted avg	0.76	0.74	0.74	2045

---runtime= 03:40:19 ---

The rest testing experiment in below all uses GPU→NVIDIA GeForce X350, Tensorflow 2.5, CUDA 11.2, CuDNN 8.1

length(number) of train set data: 6132

length(number) of validation set data: 2045

length(number) of test set data: 2045

Epoch= 25, Learning rate= 0.001

	precision	recall	f1-score	support
accuracy			0.74	2045
macro avg	0.73	0.73	0.72	2045
weighted avg	0.75	0.74	0.73	2045

---runtime= 00:44:38 ---

Using GPU

length(number) of train set data: 6132

length(number) of validation set data: 2045

length(number) of test set data: 2045

Epoch= 30, Learning rate= 0.001, batch_size: 64

	precision	recall	f1-score	support
accuracy			0.74	2045
macro avg	0.74	0.73	0.72	2045
weighted avg	0.75	0.74	0.73	2045

---runtime= 00:50:14 ---

Using GPU

length(number) of train set data: 6132

length(number) of validation set data: 2045

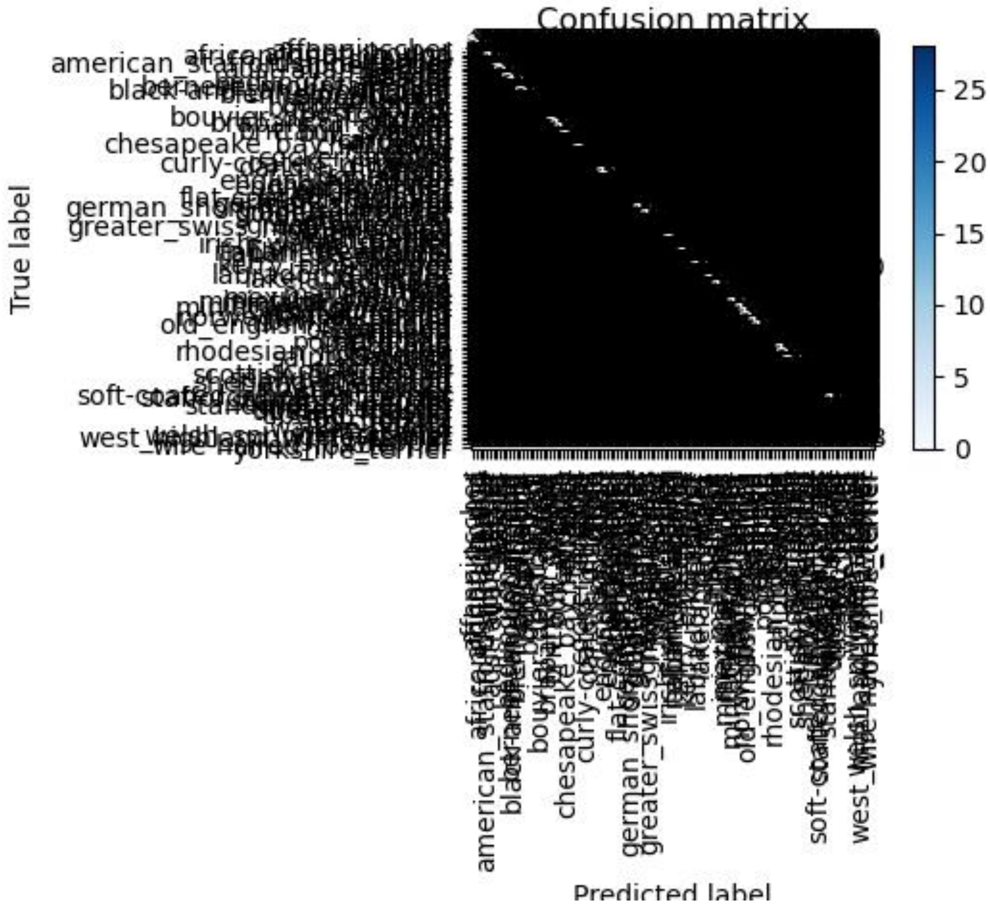
length(number) of test set data: 2045

Epoch= 20, Learning rate= 0.001, batch_size: 64

	precision	recall	f1-score	support
accuracy			0.74	2045
macro avg	0.74	0.73	0.72	2045
weighted avg	0.75	0.74	0.74	2045

---runtime= 00:35:41 ---

Below is the Confusion Matrix of this testing result.



It’s not easy to see wrong predictions in the confusion matrix graph with too many labels.

length(number) of train set data: 6132

length(number) of validation set data: 2045

length(number) of test set data: 2045

Epoch= 20, Learning rate= 1e-05, batch_size: 64 → lower learning rate

	precision	recall	f1-score	support
accuracy			0.44	2045
macro avg	0.47	0.42	0.37	2045
weighted avg	0.49	0.44	0.39	2045

---runtime= 00:34:25 ---

Learning rate too small will decrease accuracy, might stuck in local minimum.

length(number) of train set data: 6132

length(number) of validation set data: 2045

length(number) of test set data: 2045

Epoch= 20, Learning rate= 0.001, batch_size: 128

	precision	recall	f1-score	support
accuracy			0.73	2045
macro avg	0.73	0.73	0.72	2045
weighted avg	0.75	0.73	0.73	2045

---runtime= 00:36:13 ---

length(number) of train set data: 6132

length(number) of validation set data: 2045

length(number) of test set data: 2045

Epoch= 20, Learning rate= 0.001, batch_size: 32

	precision	recall	f1-score	support
accuracy			0.72	2045
macro avg	0.72	0.71	0.71	2045
weighted avg	0.74	0.72	0.72	2045

---runtime= 00:37:11 ---

Trying different train, validation, test data split.

length(number) of train set data: 9224

length(number) of validation set data: 486

length(number) of test set data: 512

Epoch= 20, Learning rate= 0.001, batch_size: 64

	precision	recall	f1-score	support
accuracy			0.74	512
macro avg	0.71	0.72	0.69	512
weighted avg	0.77	0.74	0.74	512

---runtime= 00:41:31 ---

Try different training image augmentation

train_datagen = ImageDataGenerator(rotation_range=180, width_shift_range=0.2, height_shift_range=0.2, shear_range=0.2, zoom_range=0.5, horizontal_flip=True, fill_mode='nearest')

length(number) of train set data: 6132

length(number) of validation set data: 2045

length(number) of test set data: 2045

Epoch= 20, Learning rate= 0.001, batch_size: 64

	precision	recall	f1-score	support
accuracy			0.70	2045
macro avg	0.71	0.69	0.69	2045
weighted avg	0.72	0.70	0.70	2045

---runtime= 00:36:59 ---

Below are some sample images which didn't predict correctly:

length(number) of train set data: 6132

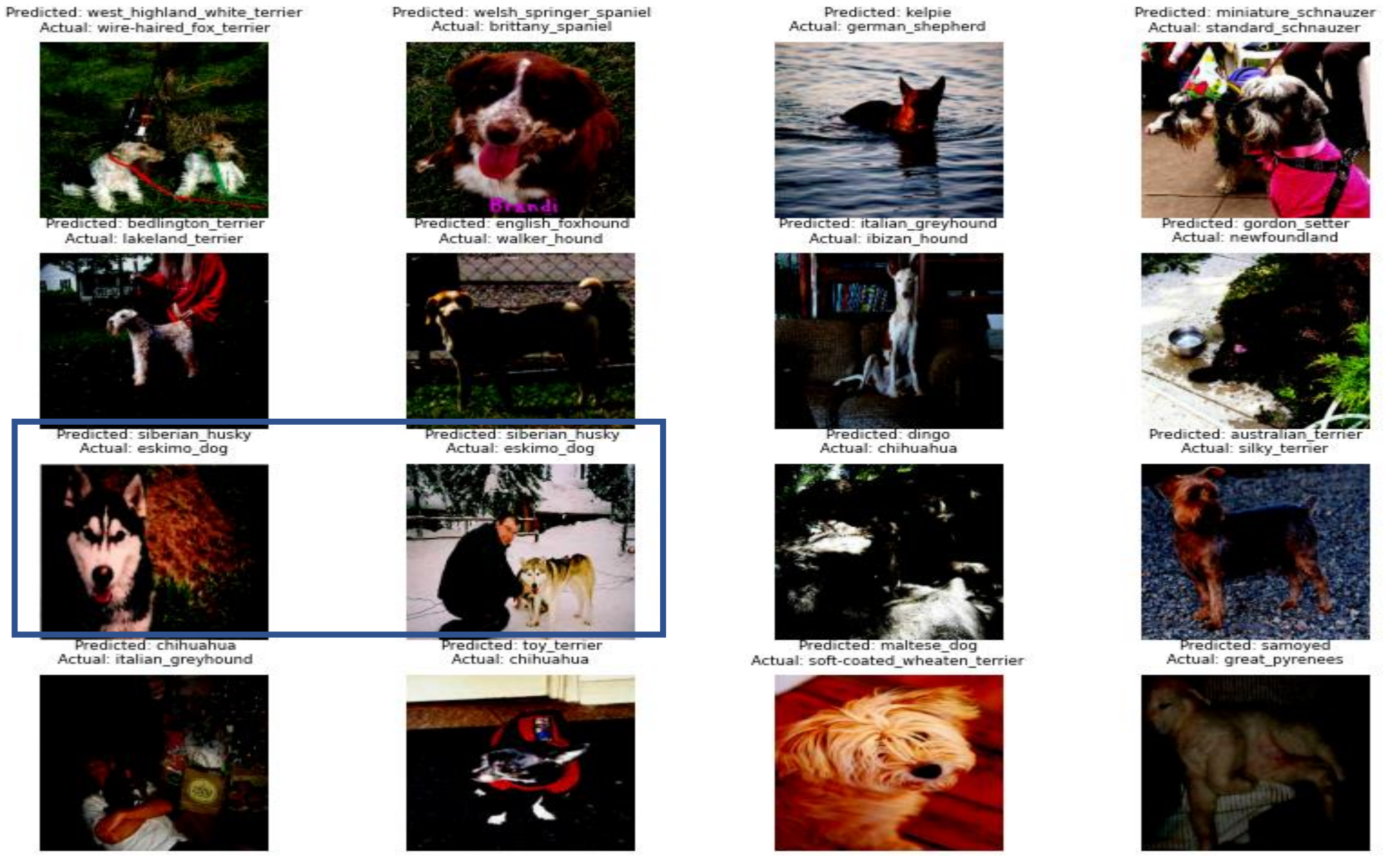
length(number) of validation set data: 2045

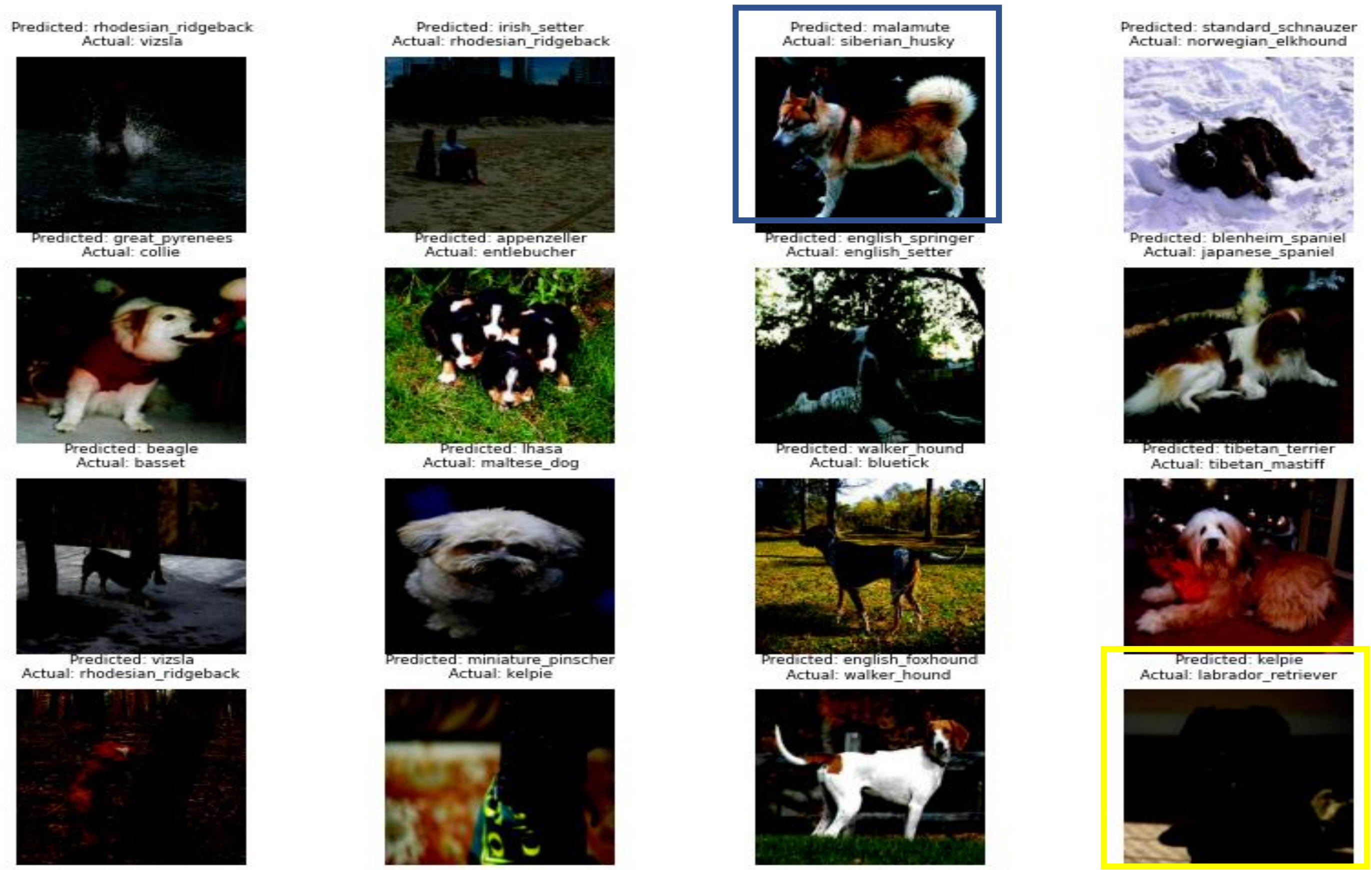
length(number) of test set data: 2045

Epoch= 20, Learning rate= 0.001, batch_size: 64

	precision	recall	f1-score	support
accuracy			0.73	2045
macro avg	0.73	0.72	0.71	2045
weighted avg	0.74	0.73	0.73	2045

---runtime= 00:36:31 ---

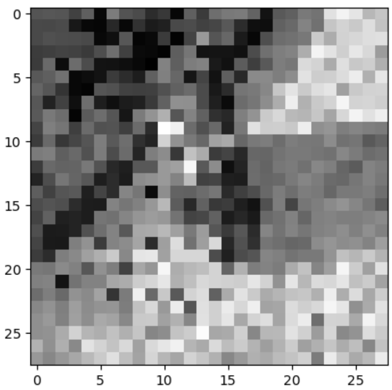
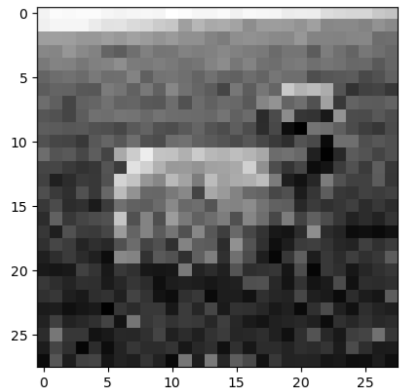
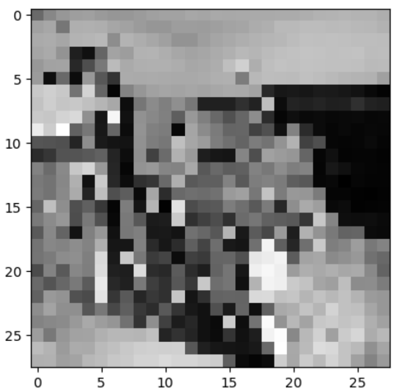
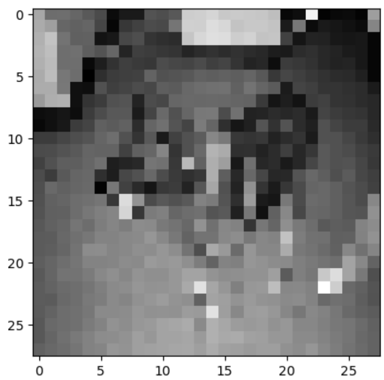




In these sample pictures which didn't predict correctly, some breeds are hard for human to distinguish. For example, the 3 blue framed images of Siberian husky, malamute or eskimo dog are mistaken. The three dog breeds are not easy for human to distinguish. Other dog breeds like [toy terrier vs chihuahua]; [maltese dog vs soft coated wheaten terrier]; [english foxhound vs walker hound], [vizsla vs Rhodesian_ridgeback]…… are also hard for human to distinguish. Also, we can see there are some images (for example: yellow framed image), that were too dark for human or the machine learning model to distinguish the dog breed.

B. Experiments of using HW4a model to train and test:

I resize the training and testing images to grayscale 28*28, (1,28,28) images, same as the Fashion-MNIST datasets. Then used the HW4a model to run training and testing.
Below are 4 sample images after doing resize and grayscale.

Below is the result.

length(number) of train set data: 8177
length(number) of test set data: 2045
Epoch= 100, Learning rate= 0.001, batch_size: 64

	precision	recall	f1-score	support
accuracy			0.05	2045
macro avg	0.05	0.05	0.04	2045
weighted avg	0.05	0.05	0.05	2045

---runtime= 00:01:44 ---

length(number) of train set data: 9710
length(number) of test set data: 512
Epoch= 20, Learning rate= 0.001, batch_size: 32

	precision	recall	f1-score	support
accuracy			0.06	512
macro avg	0.05	0.05	0.05	512
weighted avg	0.06	0.06	0.06	512

---runtime= 00:01:05 ---

Although the accuracy is quite low, but it's still a little bit better than using DummyClassifier.
Possible reasons:
1)noise from background of images (images in Fashion MNIST do not have background)
2)image size too small
3)grayscale

I tried to modify the Convolutional network, so that it can input image size 224*224:

```
class ConvNet(nn.Module):
    def __init__(self, num_classes):
        super(ConvNet, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(1, 64, kernel_size=11, stride=4, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(64, 192, kernel_size=5, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(192, 384, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
        )
        self.classifier = nn.Sequential(
            nn.Dropout(),
            nn.Linear(256 * 6 * 6, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(),
            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),
            nn.Linear(4096, num_classes),
        )
```

```
    def forward(self, x: torch.Tensor) -> torch.Tensor:
        x = self.features(x)
        x = torch.flatten(x, 1)
        x = self.classifier(x)
        return x
```

but return CUDA out of memory error:

```
Traceback (most recent call last):
  File "d:\Dpath\ASU\2021Fall\DeepLearning\Project\dog breed\dog-breed-identification\dog-breed-identification20211118a.py", line 206, in <module>
    optimizer.step()
  File "C:\Users\User\anaconda3\envs\tensorflow25\lib\site-packages\torch\optim\optimizer.py", line 88, in wrapper
    return func(*args, **kwargs)
  File "C:\Users\User\anaconda3\envs\tensorflow25\lib\site-packages\torch\autograd\grad_mode.py", line 28, in decorate_context
    return func(*args, **kwargs)
  File "C:\Users\User\anaconda3\envs\tensorflow25\lib\site-packages\torch\optim\adam.py", line 144, in step
    eps=group['eps'])
  File "C:\Users\User\anaconda3\envs\tensorflow25\lib\site-packages\torch\optim\_functional.py", line 94, in adam
    denom = (exp_avg_sq.sqrt() / math.sqrt(bias_correction2)).add_(eps)
RuntimeError: CUDA out of memory. Tried to allocate 144.00 MiB (GPU 0; 2.00 GiB total capacity; 1021.40 MiB already allocated; 0 bytes free; 1.02 GiB reserved in total by PyTorch) If reserved memory is
 >> allocated memory try setting max_split_size_mb to avoid fragmentation.  See documentation for Memory Management and PYTORCH_CUDA_ALLOC_CONF
PS D:\Dpath\ASU\2021Fall\DeepLearning\Project\dog breed\dog-breed-identification>
```

I then try to run the code on CPU, below are the results:

length(number) of train set data: 8177
length(number) of test set data: 2045
Epoch= 10, Learning rate= 0.001, batch_size: 32
Epoch [7/10], Step [100/256], Loss: 4.7445
Epoch [7/10], Step [200/256], Loss: 4.7888
Epoch [8/10], Step [100/256], Loss: 4.7834
Epoch [8/10], Step [200/256], Loss: 4.7827
Epoch [9/10], Step [100/256], Loss: 4.7784
Epoch [9/10], Step [200/256], Loss: 4.7996
Epoch [10/10], Step [100/256], Loss: 4.7538
Epoch [10/10], Step [200/256], Loss: 4.7330

	precision	recall	f1-score	support
accuracy			0.01	2045
macro avg	0.00	0.01	0.00	2045
weighted avg	0.00	0.01	0.00	2045

---runtime= 00:55:51 ---

The accuracy of the result is still low, and the Loss while training didn't go down.

I then tried using another simple CNN model which can input size 128*128 grayscale image, but still got low accuracy.
Reference website:
https://www.kaggle.com/androbomb/using-cnn-to-classify-images-w-pytorch

```
class ConvNet(nn.Module):
    def __init__(self, num_classes):
        super(ConvNet, self).__init__()
        # In the init function, we define each layer we will use in our model
        # Our images are RGB, so we have input channels = 3.
        # We will apply 12 filters in the first convolutional layer
```

```python
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=12, kernel_size=3, stride=1, padding=1)
        # A second convolutional layer takes 12 input channels, and generates 24 outputs
        self.conv2 = nn.Conv2d(in_channels=12, out_channels=24, kernel_size=3, stride=1, padding=1)
        # We in the end apply max pooling with a kernel size of 2
        self.pool = nn.MaxPool2d(kernel_size=2)
        # A drop layer deletes 20% of the features to help prevent overfitting
        self.drop = nn.Dropout2d(p=0.2)
        # Our 128x128 image tensors will be pooled twice with a kernel size of 2. 128/2/2 is 32.
        # This means that our feature tensors are now 32 x 32, and we've generated 24 of them
        # We need to flatten these in order to feed them to a fully-connected layer
        self.fc = nn.Linear(in_features=32 * 32 * 24, out_features=num_classes)

    def forward(self, x):
        # In the forward function, pass the data through the layers we defined in the init function
        # Use a ReLU activation function after layer 1 (convolution 1 and pool)
        x = F.relu(self.pool(self.conv1(x)))
        # Use a ReLU activation function after layer 2
        x = F.relu(self.pool(self.conv2(x)))
        # Select some features to drop to prevent overfitting (only drop during training)
        x = F.dropout(self.drop(x), training=self.training)
        # Flatten
        x = x.view(-1, 32 * 32 * 24)
        # Feed to fully-connected layer to predict class
        x = self.fc(x)
        # Return class probabilities via a log_softmax function
        return torch.log_softmax(x, dim=1)
```

result:

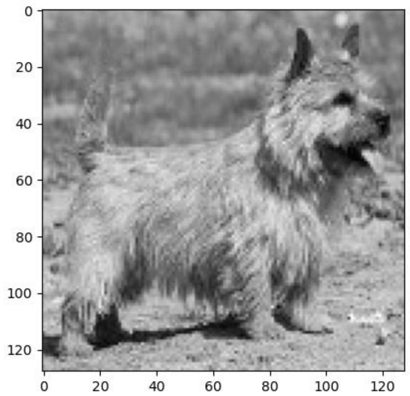
	precision	recall	f1-score	support
accuracy			0.05	2045
macro avg	0.05	0.05	0.04	2045
weighted avg	0.05	0.05	0.04	2045

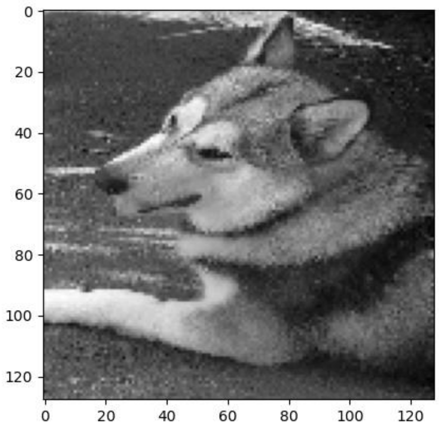
---runtime= 00:14:51 ---

However, in this experiment, the loss of training is low:

Epoch [94/100], Step [100/128], Loss: 0.0006
Epoch [95/100], Step [100/128], Loss: 0.0015
Epoch [96/100], Step [100/128], Loss: 0.0036
Epoch [97/100], Step [100/128], Loss: 0.0004
Epoch [98/100], Step [100/128], Loss: 0.0016
Epoch [99/100], Step [100/128], Loss: 0.0002
Epoch [100/100], Step [100/128], Loss: 0.0011
Test Accuracy of the model on the 10000 test images: 4.792176039119805 %

Sample images of size 128*128 grayscale:





Possible low loss low accuracy reason: High bias problem, overfitting.

To confirm whether it's due to overfitting problem, I experimented using training data to replace the testing data.

Here is the result:

Epoch [19/20], Step [100/256], Loss: 0.0281

Epoch [19/20], Step [200/256], Loss: 0.4943

Epoch [20/20], Step [100/256], Loss: 0.0747

Epoch [20/20], Step [200/256], Loss: 0.0179

Test Accuracy of the model on the 10000 test images: 99.93885288002934 %

	precision	recall	f1-score	support
accuracy			1.00	8177
macro avg	1.00	1.00	1.00	8177
weighted avg	1.00	1.00	1.00	8177

The model performs well on training data

Change the Conv2d parameters, to have more filters.

```
self.conv1 = nn.Conv2d(in_channels=1, out_channels=32, kernel_size=3, stride=1, padding=1)
self.conv2 = nn.Conv2d(in_channels=32,out_channels=64, kernel_size=3, stride=1, padding=1)
```

result:

Epoch [98/100], Step [100/205], Loss: 0.0074

Epoch [98/100], Step [200/205], Loss: 0.0372

Epoch [99/100], Step [100/205], Loss: 0.3456

Epoch [99/100], Step [200/205], Loss: 0.0103

Epoch [100/100], Step [100/205], Loss: 0.3318

Epoch [100/100], Step [200/205], Loss: 0.0133

Test Accuracy of the model on the 10000 test images: 3.374083129584352 %

	precision	recall	f1-score	support
accuracy			0.03	2045

macro avg	0.04	0.03	0.03	2045
weighted avg	0.04	0.03	0.03	2045

---runtime= 00:25:51 ---

Although runtime increased, accuracy does not improve.

I then tried to use the concept of validation dataset while training the model. I evaluated and tested the model with different epoch of training, to see when I should stop the training. However, the accuracy did not get higher than 5% in any epochs. Below are the result of trying different epochs.

length(number) of train set data: 8177
length(number) of test set data: 2045
Learning rate= 0.001, batch_size: 32

[epoch: 1, accuracy: 1.36919315403423 %]
[epoch: 5, accuracy: 4.743276283618582 %]
[epoch: 10, accuracy: 5.036674816625917 %]
[epoch: 15, accuracy: 4.205378973105135 %]
[epoch: 20, accuracy: 3.8141809290953543 %]
[epoch: 30, accuracy: 4.205378973105135 %]
[epoch: 40, accuracy: 4.30317848410758 %]
[epoch: 50, accuracy: 4.69437652811736 %]
[epoch: 60, accuracy: 4.743276283618582 %]
[epoch: 70, accuracy: 4.400977995110025 %]
[epoch: 80, accuracy: 5.330073349633252 %]
[epoch: 90, accuracy: 4.156479217603912 %]
[epoch: 100, accuracy: 4.352078239608802 %]

The below github repository contain my programs for this project:
https://github.com/KuoChiaCheng0318/ASU_CSE598_IntroDeepLearning_Project