Intelligent World Informatics Lecture V
Final Project

Paul Lödige
Student ID: 37-229753
paul.loedige@student.kit.edu

February 3, 2023

1 Assignment

- Define your own project
  - Describe the task and the goal you would like to achieve
  - Experiment on any public dataset provided by Tensorflow:
    - Full list: `https://www.tensorflow.org/datasets/catalog/overview`
    - Simple datasets: `https://www.tensorflow.org/api_docs/python/tf/keras/datasets`
  - Do one of the followings:
    1. Apply two different deep learning techniques you learned from this class
       - e.g.: Comparing Dense DNN vs RNN method on the same dataset (what pros and cons?)
       - e.g.: Combining CNN and RNN on a dataset (why the combination is preferable?)
    2. Apply transfer learning with two different base models
       - Explain the reasons why the base models are reasonable choices.
       - Survey on the internet with keywords such as "best model for imagenet/cifar10/mnist", "tensorflow pre-trained models", etc, to find the name of the model
       - Download the base model through tensorflow
         - Just like in the code, simply change the name of the base model. List of available base models in Tensorflow:
           - `https://www.tensorflow.org/api_docs/python/tf/keras/applications`
         - Add layers, dropouts, use different learning rate, epoch, batch_size, etc.

1.1 Additional Details

- Write 1 $\sim$ 2 page(s) (excluding the code) describing your project:
  - Imagine the project as a kind of "mini research paper"
  - What is the problem? What is your motivation?
  - How are you solving them? What deep learning algorithms?
  - Outline the experiment and test on the dataset.
  - What are the results?
    - Add at least 1 figure and insert your analysis
  - Attach the code at the end.
  - Include your name, affiliation, student number, as well as your university e-mail address in case if we need to contact you for clarification
  - Put all the above in a single PDF and upload to ITC-LMS

2 Project Goal

This project aims to compare different deep learning techniques. The goal is to discern the pros and cons of the different techniques when it comes to their use on character recognitions in images. This approach was chosen because it doesn't focus on the learning of object concepts but rather on pattern recognition. Since pattern recognition and character recognition in particular is a less complex topic than concept classification it allows for the use of less complex models which are easier to understand.

2.1 The Dataset

The project uses the "svhn_cropped" dataset (`https://www.tensorflow.org/datasets/catalog/svhn_cropped`). The Street View House Numbers (SVHN) dataset was created at Stanford University and includes "73 257 digits for training, 26 032 digits for testing, and 531 131 additional, somewhat less difficult samples, to use as extra training data" (`http://ufldl.stanford.edu/housenumbers/`). The dataset provided by TensorFlow uses the "MNIST-like 32-by-32 images centered around a single character" version.

3 Methods

3.1 Dense Neural Network

The first method tried to find a good classification model was to use regular Dense Neural Networks. Through experimentation it was determined that the classification accuracy only varied slightly when using a different amount of hidden layers (see appendix A.1).

3.2 Convolutional Neural Network (CNN)

When it comes to image classification CNNs are often used as they reduce the dimensionality of the input. This in turn simplifies the problem and makes it easier to classify. Multiple configurations for CNNs where tried (see appendix A.2) The best CNN found during experimentation had the following configuration:

```
configs['convolutional neural network (3 convolutions)'] = {
        'model': models.Sequential([
            layers.Conv2D(32, (3,3), activation='relu', input_shape=(32,32,3)),
            layers.MaxPooling2D((2,2)),
            layers.Conv2D(64, (3,3), activation='relu'),
            layers.MaxPooling2D((2,2)),
            layers.Conv2D(128, (3,3), activation='relu'),
            layers.MaxPooling2D((2,2)),
            layers.Flatten(),
            layers.Dense(64, activation='relu'),
            layers.Dense(64, activation='relu'),
            layers.Dense(10, activation='softmax')
        ]),
        'optimizer': 'adam',
        'loss': 'sparse_categorical_crossentropy',
        'metrics': ['accuracy'],
        'epochs': 10,
        'batch_size': 128
    }
```

4 Comparison

While the Dense Neural Networks can be trained efficiently trained (a few seconds per epoch), they are also less accurate than the CNNs. As can be seen in fig. 1 the Dense Neural Networks achieved a validation accuracy of around 70 %. The CNNs on the other hand achieved around 80 – 85 %.

It can also be seen that the type of model has a bigger impact on the accuracy than its complexity. Even the CNN with just one convolutional layer performs better than the Dense neural network with 5 hidden dense layers.

4.1 VGG16

As can be seen from the comparison of the two previous methods CNNs hold a clear advantage when it comes to image classification. In order to find an even better model for classification it therefore makes sense to look at CNNs developed by other researchers. VGG16 is one CNN that is commonly used for these kind of tasks and is also available as part of the texttttensorflow.keras library. While the attempt at transfer learning was unsuccessful (see appendix A.3), training the whole model pre-trained on the "Imagenet" dataset with a learning rate of 0.001 produced a model with even higher accuracy than the CNNs mentioned before. While the training of this model is very time intensive it gave a validation accuracy of over 92 % (fig. 1).

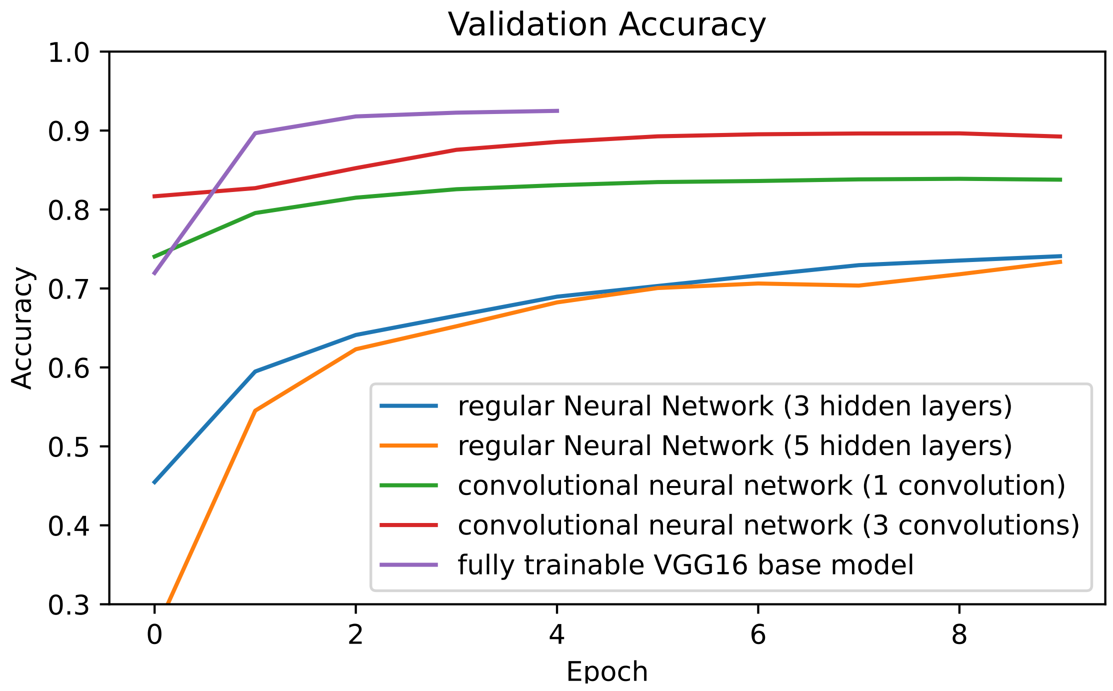


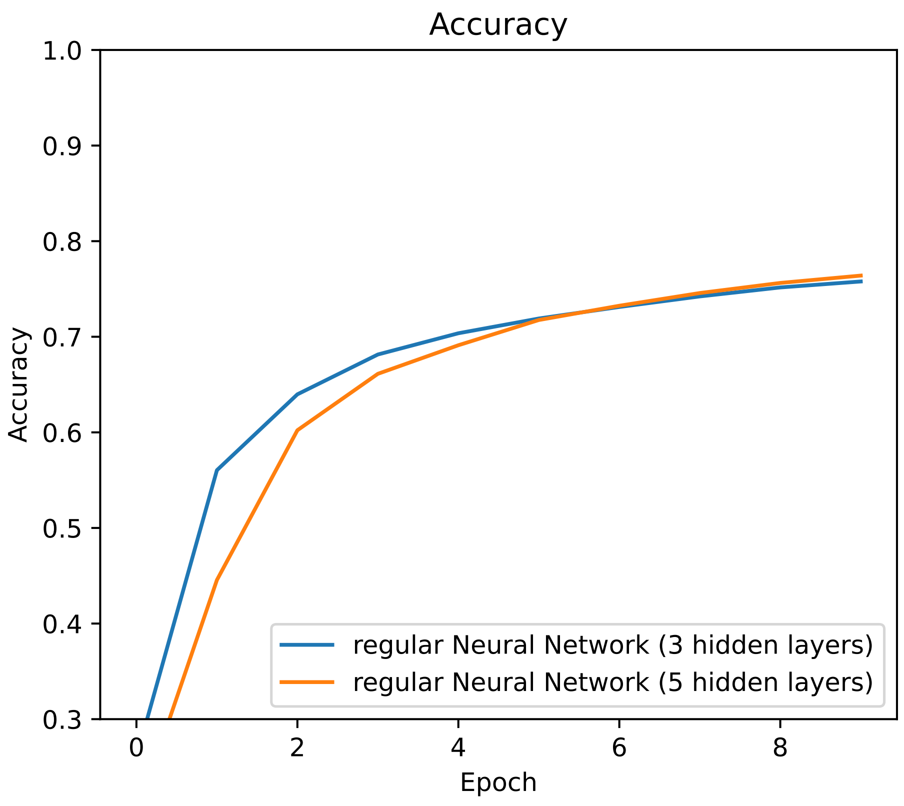
Figure 1: Validation Accuracy of the different models

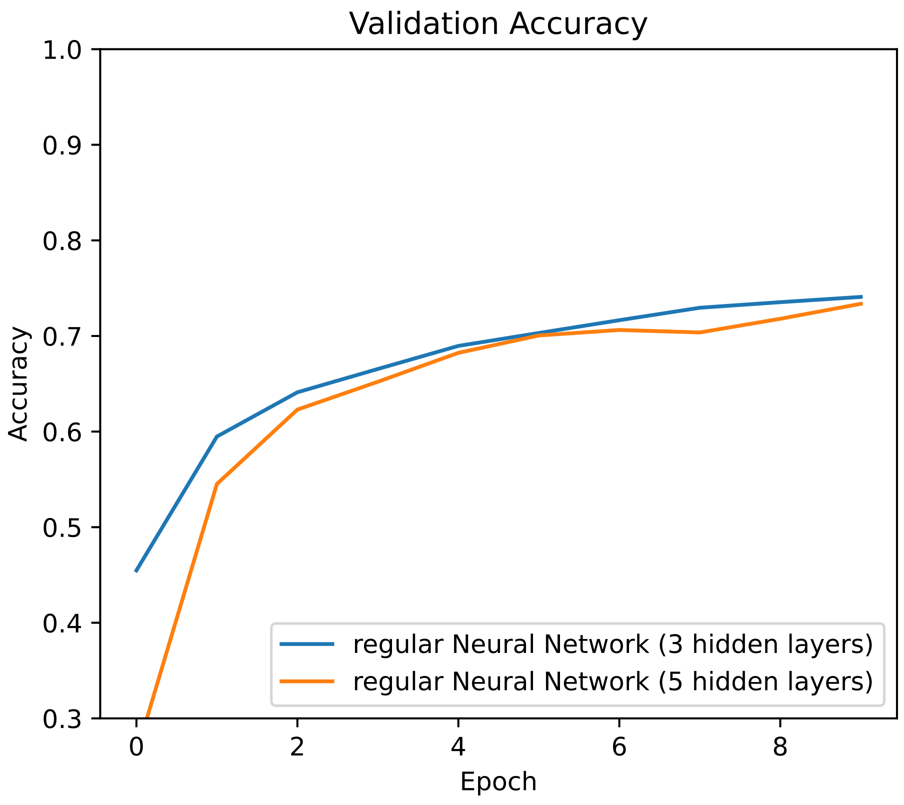
5 Summary

In this project it has been shown that Convolutional Neural Networks outperform Dense Neural Networks when it comes to classifying the SVHN dataset. Additionally, if the goal is to achieve the highest accuracy possible one should use a pre-existing model designed for solving problems in a similar context.

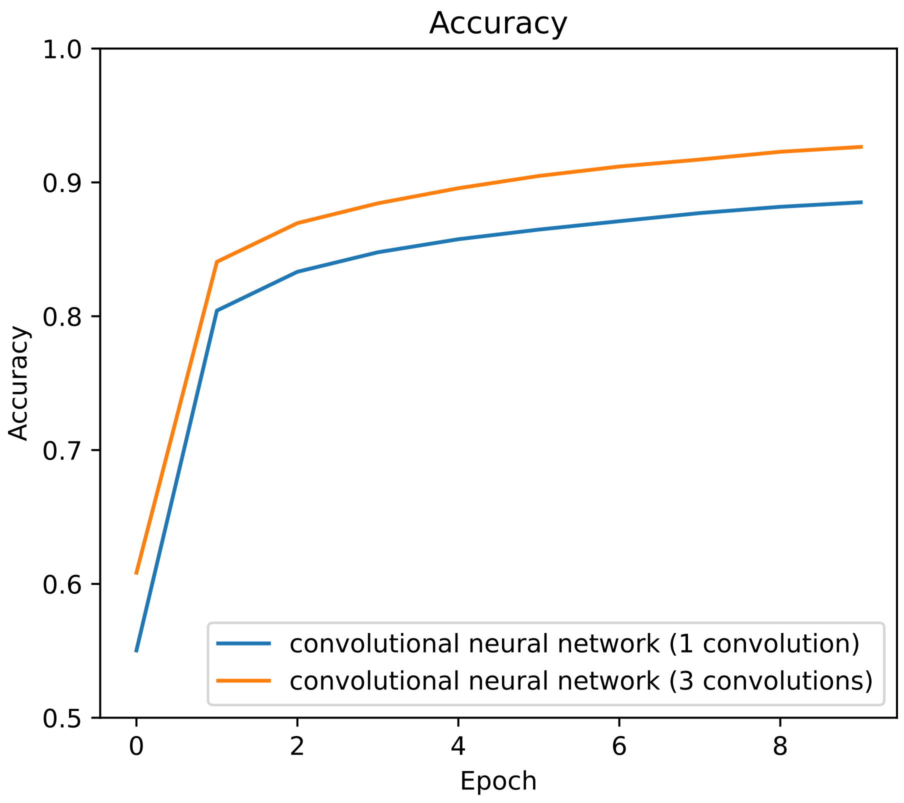
A Appendix

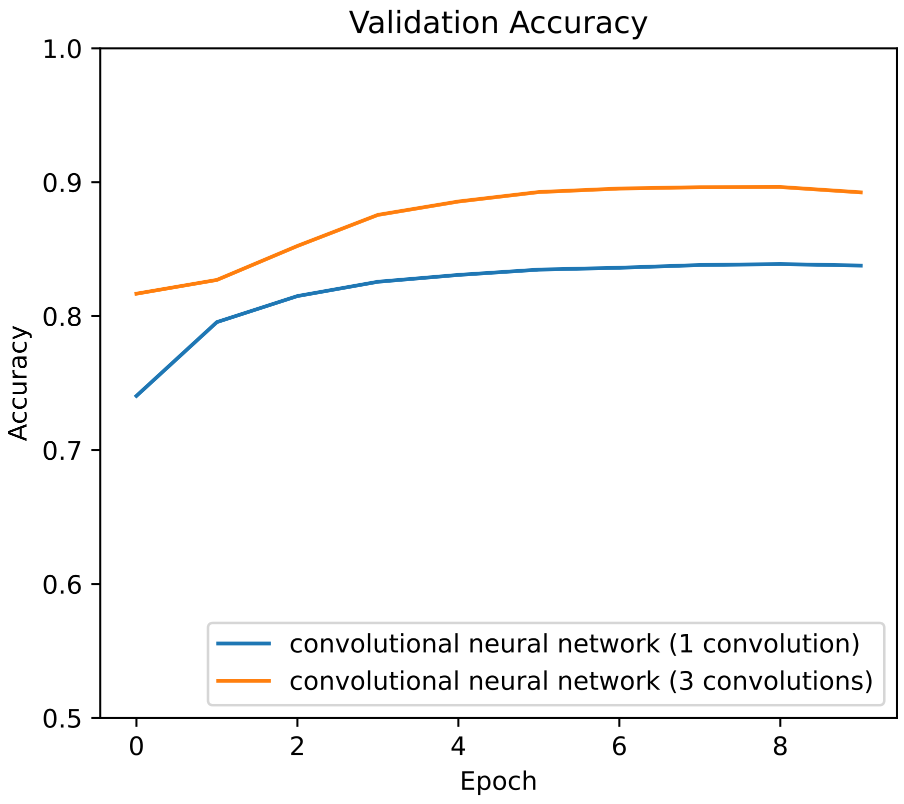
A.1 Dense Neural Network Comparison



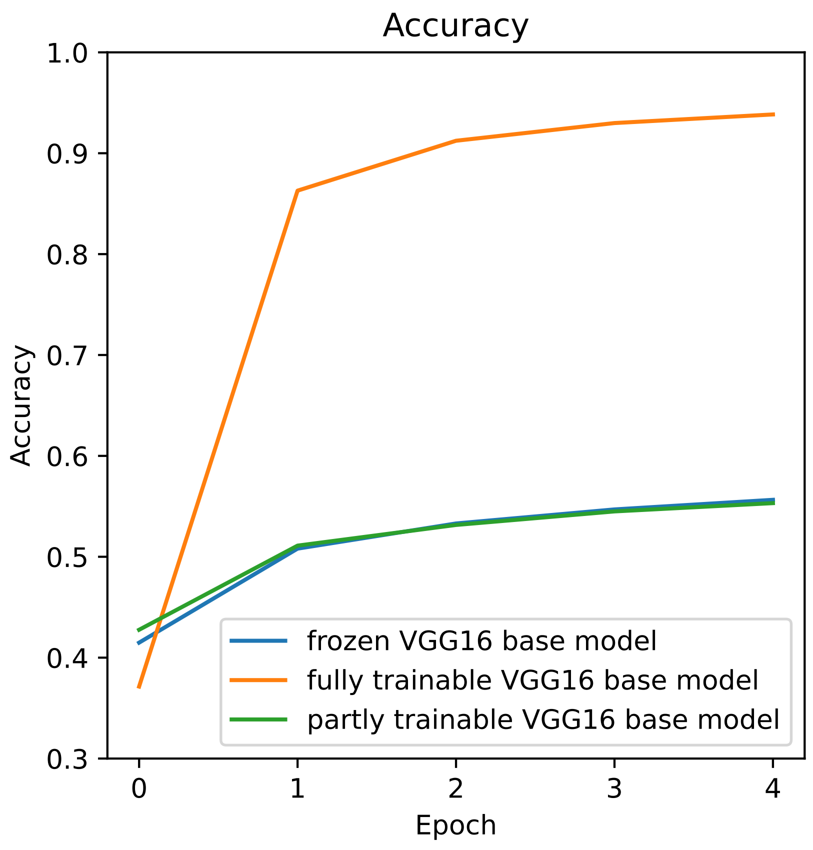


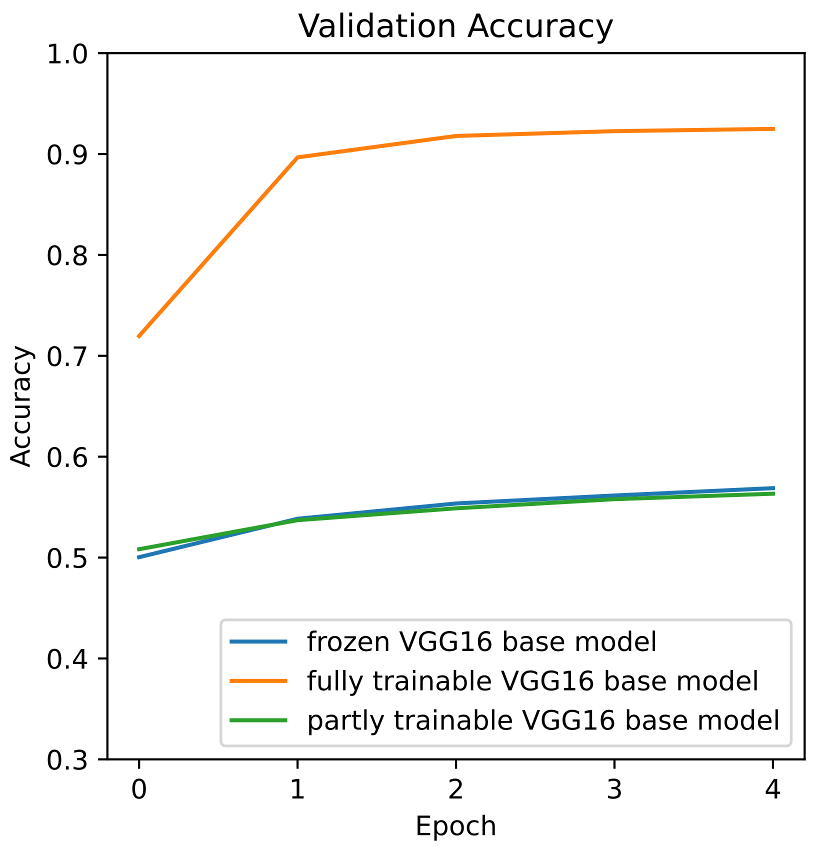
A.2 CNN Comparison





A.3 Transfer Learning VGG16





A.4 Code

The following code is an exported and slightly formatted version of a Jupyter Notebook that is also available at `https://git.ploedige.com/Intelligent_World_Informatics_V/Final_Project`.

```python
# Import libraries
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import layers, models, applications, optimizers
import tensorflow_datasets as tfds
import tkinter.messagebox # for notifications

# Load Dataset
# https://www.tensorflow.org/datasets/keras_example
# split into training and test data
ds_train, ds_test = tfds.load(
    'svhn_cropped',
    split=['train', 'test'],
    shuffle_files=True,
    as_supervised=True
    )

# Confirm Data
for image, label in ds_train.take(1):
    plt.imshow(image)
    plt.show()

# Define normalization function
def normalize_img(image, label):
    return tf.cast(image, tf.float32) / 255.0, label

# Build a training pipeline
ds_train = ds_train.map(normalize_img, num_parallel_calls=tf.data.AUTOTUNE)
ds_train = ds_train.cache()
ds_train = ds_train.batch(128)
ds_train = ds_train.prefetch(tf.data.AUTOTUNE)

# Build an evaluation pipeline
ds_test = ds_test.map(normalize_img, num_parallel_calls=tf.data.AUTOTUNE)
ds_test = ds_test.cache()
ds_test = ds_test.batch(128)
ds_test = ds_test.prefetch(tf.data.AUTOTUNE)

# Configurations
configs = {}

configs['regular Neural Network (3 hidden layers)'] = {
        'model': models.Sequential([
            layers.Flatten(input_shape=(32, 32, 3)),
            layers.Dense(256, activation='relu'),
            layers.Dense(128, activation='relu'),
            layers.Dense(64, activation='relu'),
            layers.Dense(10, activation='softmax')
        ]),
        'optimizer': 'adam',
        'loss': 'sparse_categorical_crossentropy',
        'metrics': ['accuracy'],
        'epochs': 30,
        'batch_size': 64
    }

configs['regular Neural Network (5 hidden layers)'] = {
        'model': models.Sequential([
            layers.Flatten(input_shape=(32, 32, 3)),
            layers.Dense(1024, activation='relu'),
            layers.Dense(512, activation='relu'),
            layers.Dense(256, activation='relu'),
            layers.Dense(128, activation='relu'),
            layers.Dense(64, activation='relu'),
            layers.Dense(10, activation='softmax')
        ]),
        'optimizer': 'adam',
        'loss': 'sparse_categorical_crossentropy',
        'metrics': ['accuracy'],
        'epochs': 30,
        'batch_size': 64
    }

configs['convolutional neural network (1 convolution)'] = {
        'model': models.Sequential([
            layers.Conv2D(32, (3,3), activation='relu', input_shape=(32,32,3)),
            layers.MaxPooling2D((2,2)),
            layers.Flatten(),
            layers.Dense(64, activation='relu'),
            layers.Dense(64, activation='relu'),
            layers.Dense(10, activation='softmax')
        ]),
        'optimizer': 'adam',
        'loss': 'sparse_categorical_crossentropy',
        'metrics': ['accuracy'],
        'epochs': 10,
        'batch_size': 64
    }

configs['convolutional neural network (3 convolutions)'] = {
        'model': models.Sequential([
            layers.Conv2D(32, (3,3), activation='relu', input_shape=(32,32,3)),
            layers.MaxPooling2D((2,2)),
            layers.Conv2D(64, (3,3), activation='relu'),
            layers.MaxPooling2D((2,2)),
            layers.Conv2D(128, (3,3), activation='relu'),
            layers.MaxPooling2D((2,2)),
            layers.Flatten(),
            layers.Dense(64, activation='relu'),
            layers.Dense(64, activation='relu'),
            layers.Dense(10, activation='softmax')
        ]),
        'optimizer': 'adam',
        'loss': 'sparse_categorical_crossentropy',
        'metrics': ['accuracy'],
        'epochs': 10,
        'batch_size': 128
    }

vgg16_base_model = applications.VGG16(input_shape=(32,32,3), include_top= False)
# freeze the VGG16 model
vgg16_base_model.trainable = False
configs['frozen VGG16 base model'] = {
        'model': models.Sequential([
            vgg16_base_model,
            layers.Flatten(),
            layers.Dense(64, activation='relu'),
            layers.Dense(32, activation='relu'),
            layers.Dense(10, activation='softmax'),
        ]),
        'optimizer': 'adam',
        'loss': 'sparse_categorical_crossentropy',
        'metrics': ['accuracy'],
        'epochs': 5,
        'batch_size': 64
    }

vgg16_base_model = applications.VGG16(input_shape=(32,32,3), include_top= False)
configs['fully trainable VGG16 base model'] = {
        'model': models.Sequential([
            vgg16_base_model,
            layers.Flatten(),
            layers.Dense(64, activation='relu'),
            layers.Dense(32, activation='relu'),
            layers.Dense(10, activation='softmax'),
        ]),
        'optimizer': optimizers.Adam(learning_rate=0.001),
        'loss': 'sparse_categorical_crossentropy',
        'metrics': ['accuracy'],
        'epochs': 5,
        'batch_size': 64
    }

vgg16_base_model = applications.VGG16(input_shape=(32,32,3), include_top= False)
# freeze the VGG16 model
```

```python
vgg16_base_model.trainable = False
for layer in vgg16_base_model.layers[-6:]:
    layer.trainable=True

configs['partly trainable VGG16 base model'] = {
        'model': models.Sequential([
            vgg16_base_model,
            layers.Flatten(),
            layers.Dense(64, activation='relu'),
            layers.Dense(32, activation='relu'),
            layers.Dense(10, activation='softmax'),
        ]),
        'optimizer': optimizers.Adam(learning_rate=0.001),
        'loss': 'sparse_categorical_crossentropy',
        'metrics': ['accuracy'],
        'epochs': 5,
        'batch_size': 64
    }

# Compile the Models
for config in configs.values():
    if 'history' not in config.keys():
        config['model'].compile(
            optimizer=config['optimizer'],
            loss=config['loss'],
            metrics=config['metrics']
        )

# Train and Evaluate the Models
try:
    for config_name, config in configs.items():
        if 'history' in config.keys():
            print(f'Already trained model "{config_name}"')
        else:
            print(f'Now training model "{config_name}"')
            config['history'] = config['model'].fit(ds_train, epochs=config['epochs'], validation_data=ds_test, batch_size=config['batch_size'])#, verbose=0)
except Exception as e:
    print(e)
    tkinter.messagebox.showerror("ERROR", f"ERROR: {e}")

# Plot the accuracy
plt.figure(figsize=(10,5))
for config_name, config in configs.items():
    plt.plot(config['history'].history['accuracy'], label=config_name)
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
# plt.ylim([0.75, 1])
plt.legend(loc='lower right')
plt.title("Accuracy")
plt.show()

# Plot the validation accuracy
plt.figure(figsize=(10,5))
for config_name, config in configs.items():
    plt.plot(config['history'].history['val_accuracy'], label=config_name)
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
# plt.ylim([0.8, 1])
plt.legend(loc='lower right')
plt.title("Validation Accuracy")
plt.show()

# Notify when done
tkinter.messagebox.showinfo("DONE", "DONE")
```