CDS Machine Learning Week 5 Assignment - Image Classification

By Jasper Pieterse, Daria Mihalia and Jochem Pannekoek

The goal of this assignment is to test and compare some simple deep learning architectures for the problem of image classification. We will be using Tensorflow and the CIFAR-10 dataset. We re-cycled code and markdown from This tutorial and this Jupyter notebook.

```
from __future__ import absolute_import, division, print_function, unicode_literals
import datetime
import time
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import datasets, layers, models

#seeds for reproducibility
np.random.seed(0)
tf.random.set_seed(0)
```

Download and prepare the CIFAR10 dataset

The CIFAR10 dataset contains 60,000 color images in 10 classes, with 6,000 images in each class. The dataset is divided into 50,000 training images and 10,000 testing images. The classes are mutually exclusive and there is no overlap between them.

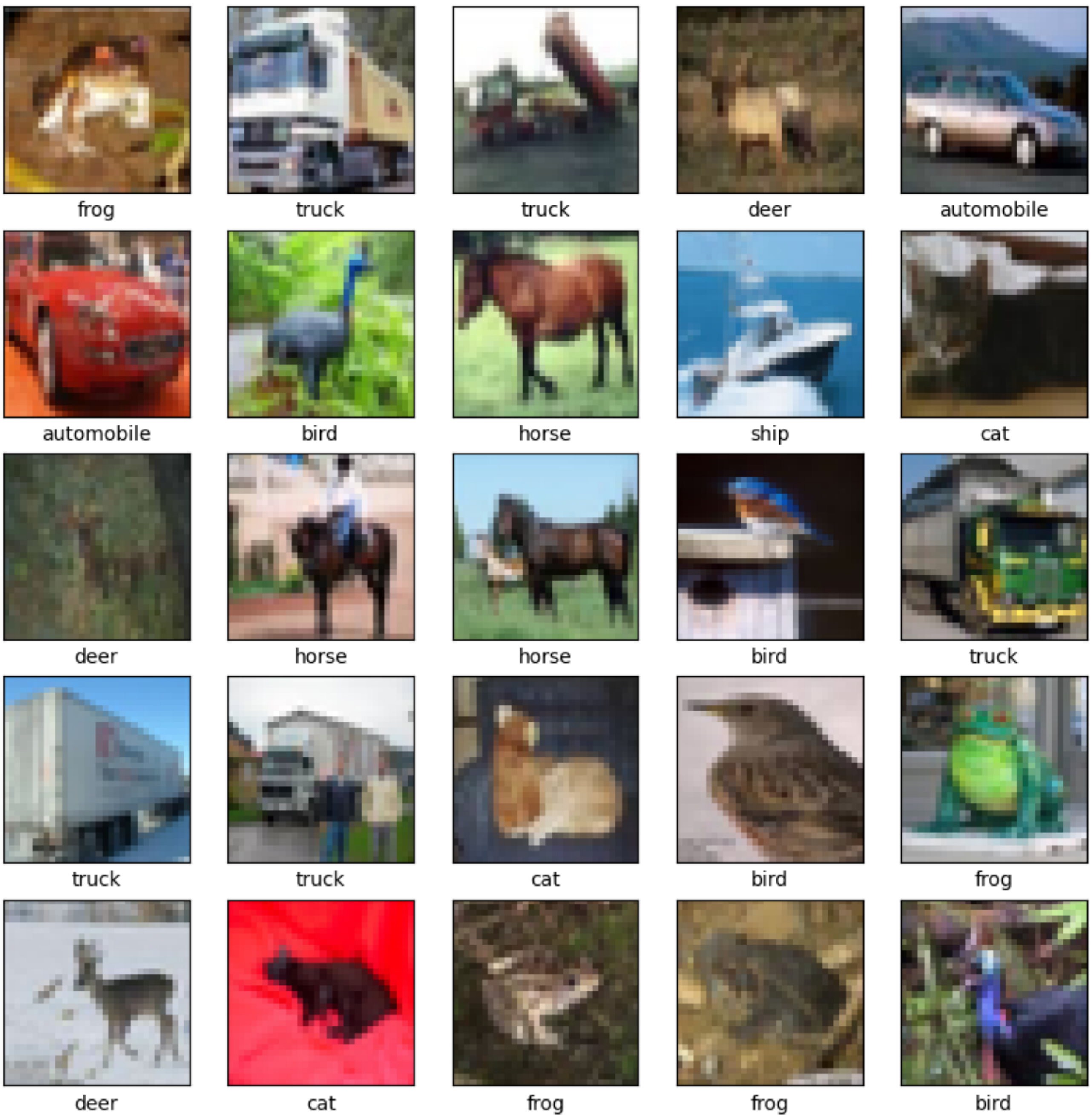
```
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_da

# normalize pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 2s 0us/step
```

Verify the data

To verify that the dataset looks correct, let's plot the first 25 images from the training set and display the class name below each image:

```
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']

plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i])
    # The CIFAR labels happen to be arrays,
    # which is why you need the extra index
    plt.xlabel(class_names[train_labels[i][0]])
plt.show()
```



Exercise 1 - Multi Layer Perceptron (MLP)

Modify the provided script 'perceptron.py' to build a MLP. Use architectures with 0, 1 and 2 hidden layers. Keep the complexity of the model bounded so runs do not take much more than 1 hour to reach the maximum of testing accuracy. Notice that the input needs to be "flattened" since there is no spatial structure in this fully connected design. This can be achieved by adding a dummy layer with no free parameters with "layers.Flatten()" as the first layer in the constructor "model.Sequential()". Obtain the learning curves and discuss the results. Report the optimizer in use, initialization parameters, the learning rate, etc. Is early stopping convenient in this model?

Defining the models

```
#The provided model in perceptron.py is a MLP with no hidden layers.
#To make it a multi-layer perceptron, we simply add 1 or 2 hidden layers.

model_0 = models.Sequential([  #architecture constructor
        layers.Flatten(),
        layers.Dense(10, activation='softmax')])

model_1 = models.Sequential([
    layers.Flatten(),                         # input layer [flatten image to 1D vecto
    layers.Dense(512, activation='relu'), # hidden layer with 512 neurons and relu
    layers.Dense(10, activation='softmax')# output layer of 10 catergories with so
])

model_2 = models.Sequential([
    layers.Flatten(),
    layers.Dense(512, activation='relu'),
    layers.Dense(512, activation='relu'), # add 2nd hidden layer
    layers.Dense(10, activation='softmax')
])

#use same optimizer and loss function for all models
opt = tf.keras.optimizers.legacy.Adam(learning_rate=0.001)
np.random.seed(0)
tf.random.set_seed(0)
```

```
model_0.compile(optimizer= opt,
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model_1.compile(optimizer= opt,
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model_2.compile(optimizer= opt,
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

Training the models

```
models_list = [model_0, model_1, model_2]
histories_without_es = []  # store history of each model without early stopping
histories_with_es = []     # store history of each model with early stopping
print_statements = []      # store print to print at the end

# train each model and plot
for i, model in enumerate(models_list):

    # train the model without early stopping [callback argument]
    print(f"Training model_{i} without early stopping...")
    history_without_es = model.fit(train_images, train_labels, epochs=50,
                            validation_data=(test_images, test_labels))
    # evaluate and store
    histories_without_es.append(history_without_es)
    test_loss, test_acc = model.evaluate(test_images, test_labels)
    print_statement = f"Model_{i} without early stopping - Test Loss: {test_loss},
    print_statements.append(print_statement)
    print(print_statement)

    # re-compile the model to ensure it is the same as the original
    model.compile(optimizer= opt, loss='sparse_categorical_crossentropy', metrics=[

    # train again but now with early stopping
    print(f"Training model_{i} with early stopping...")
    callback = tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=3)
    history_with_es = model.fit(train_images, train_labels, epochs=50,
                               validation_data=(test_images, test_labels), call
    # evaluate and store
    histories_with_es.append(history_with_es)
    test_loss, test_acc = model.evaluate(test_images, test_labels)
    print_statement = f"Model_{i} with early stopping - Test Loss: {test_loss}, Tes
    print_statements.append(print_statement)
    print(print_statement)
```

```
# print all statements again at the end
print("\n" + "="*30 + " Print Statements " + "="*30)
for statement in print_statements:
    print(statement)

# plot learning curves
def plot_learning_curves(histories, title):
    for i, history in enumerate(histories):
        plt.figure(figsize=(12, 6))

        plt.subplot(1, 2, 1)
        plt.plot(history.history['accuracy'], label='Train Accuracy', color = 'Medi
        plt.plot(history.history['val_accuracy'], label='Validation Accuracy', colo
        plt.title(f'Model_{i} - {title} - Accuracy')
        plt.xlabel('Epoch')
        plt.ylabel('Accuracy')
        plt.legend()

        plt.subplot(1, 2, 2)
        plt.plot(history.history['loss'], label='Train Loss', color = 'DarkOrange')
        plt.plot(history.history['val_loss'], label='Validation Loss', color = 'Ora
        plt.title(f'Model_{i} - {title} - Loss')
        plt.xlabel('Epoch')
        plt.ylabel('Loss')
        plt.legend()

        plt.show()

plot_learning_curves(histories_with_es, 'With early stopping')
plot_learning_curves(histories_without_es, 'Without early stopping')
```

```
Training model_0 without early stopping...
Epoch 1/50
1563/1563 [==============================] - 7s 4ms/step - loss: 1.7154 - accu
Epoch 2/50
1563/1563 [==============================] - 4s 3ms/step - loss: 1.7207 - accu
Epoch 3/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7180 - accu
Epoch 4/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7113 - accu
Epoch 5/50
1563/1563 [==============================] - 4s 3ms/step - loss: 1.7249 - accu
Epoch 6/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7106 - accu
Epoch 7/50
1563/1563 [==============================] - 4s 3ms/step - loss: 1.7076 - accu
Epoch 8/50
1563/1563 [==============================] - 4s 3ms/step - loss: 1.7135 - accu
Epoch 9/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7173 - accu
```

```
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7175 - accu
Epoch 10/50
1563/1563 [==============================] - 4s 3ms/step - loss: 1.7186 - accu
Epoch 11/50
1563/1563 [==============================] - 6s 4ms/step - loss: 1.7126 - accu
Epoch 12/50
1563/1563 [==============================] - 4s 3ms/step - loss: 1.7131 - accu
Epoch 13/50
1563/1563 [==============================] - 4s 3ms/step - loss: 1.7176 - accu
Epoch 14/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7163 - accu
Epoch 15/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7155 - accu
Epoch 16/50
1563/1563 [==============================] - 7s 5ms/step - loss: 1.7116 - accu
Epoch 17/50
1563/1563 [==============================] - 4s 3ms/step - loss: 1.7074 - accu
Epoch 18/50
1563/1563 [==============================] - 4s 3ms/step - loss: 1.7039 - accu
Epoch 19/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7168 - accu
Epoch 20/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7036 - accu
Epoch 21/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7098 - accu
Epoch 22/50
1563/1563 [==============================] - 4s 3ms/step - loss: 1.7118 - accu
Epoch 23/50
1563/1563 [==============================] - 4s 3ms/step - loss: 1.7198 - accu
Epoch 24/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7129 - accu
Epoch 25/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7134 - accu
Epoch 26/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7071 - accu
Epoch 27/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7093 - accu
Epoch 28/50
1563/1563 [==============================] - 4s 3ms/step - loss: 1.7097 - accu
Epoch 29/50
1563/1563 [==============================] - 6s 4ms/step - loss: 1.7108 - accu
Epoch 30/50
1563/1563 [==============================] - 4s 3ms/step - loss: 1.7111 - accu
Epoch 31/50
1563/1563 [==============================] - 4s 3ms/step - loss: 1.7112 - accu
Epoch 32/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7129 - accu
Epoch 33/50
1563/1563 [==============================] - 4s 3ms/step - loss: 1.7152 - accu
Epoch 34/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7093 - accu
Epoch 35/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7044 - accu
```

```
Epoch 36/50
1563/1563 [==============================] - 4s 3ms/step - loss: 1.7066 - accu
Epoch 37/50
1563/1563 [==============================] - 6s 4ms/step - loss: 1.7052 - accu
Epoch 38/50
1563/1563 [==============================] - 4s 3ms/step - loss: 1.7099 - accu
Epoch 39/50
1563/1563 [==============================] - 4s 3ms/step - loss: 1.7057 - accu
Epoch 40/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7034 - accu
Epoch 41/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7019 - accu
Epoch 42/50
1563/1563 [==============================] - 6s 4ms/step - loss: 1.7067 - accu
Epoch 43/50
1563/1563 [==============================] - 4s 3ms/step - loss: 1.6974 - accu
Epoch 44/50
1563/1563 [==============================] - 4s 3ms/step - loss: 1.6952 - accu
Epoch 45/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7055 - accu
Epoch 46/50
1563/1563 [==============================] - 4s 3ms/step - loss: 1.7035 - accu
Epoch 47/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.6938 - accu
Epoch 48/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7088 - accu
Epoch 49/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.6975 - accu
Epoch 50/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.6987 - accu
313/313 [==============================] - 1s 2ms/step - loss: 2.0098 - accura
Model_0 without early stopping - Test Loss: 2.009751081466675, Test Accuracy:
Training model_0 with early stopping...
Epoch 1/10
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7008 - accu
Epoch 2/10
1563/1563 [==============================] - 4s 3ms/step - loss: 1.7067 - accu
Epoch 3/10
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7041 - accu
Epoch 4/10
1563/1563 [==============================] - 4s 3ms/step - loss: 1.6975 - accu
Epoch 5/10
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7109 - accu
Epoch 6/10
1563/1563 [==============================] - 5s 3ms/step - loss: 1.6969 - accu
313/313 [==============================] - 1s 2ms/step - loss: 1.9691 - accura
Model_0 with early stopping - Test Loss: 1.9690543413162231, Test Accuracy: 0
Training model_1 without early stopping...
Epoch 1/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.5078 - accu
Epoch 2/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.5079 - accu
```

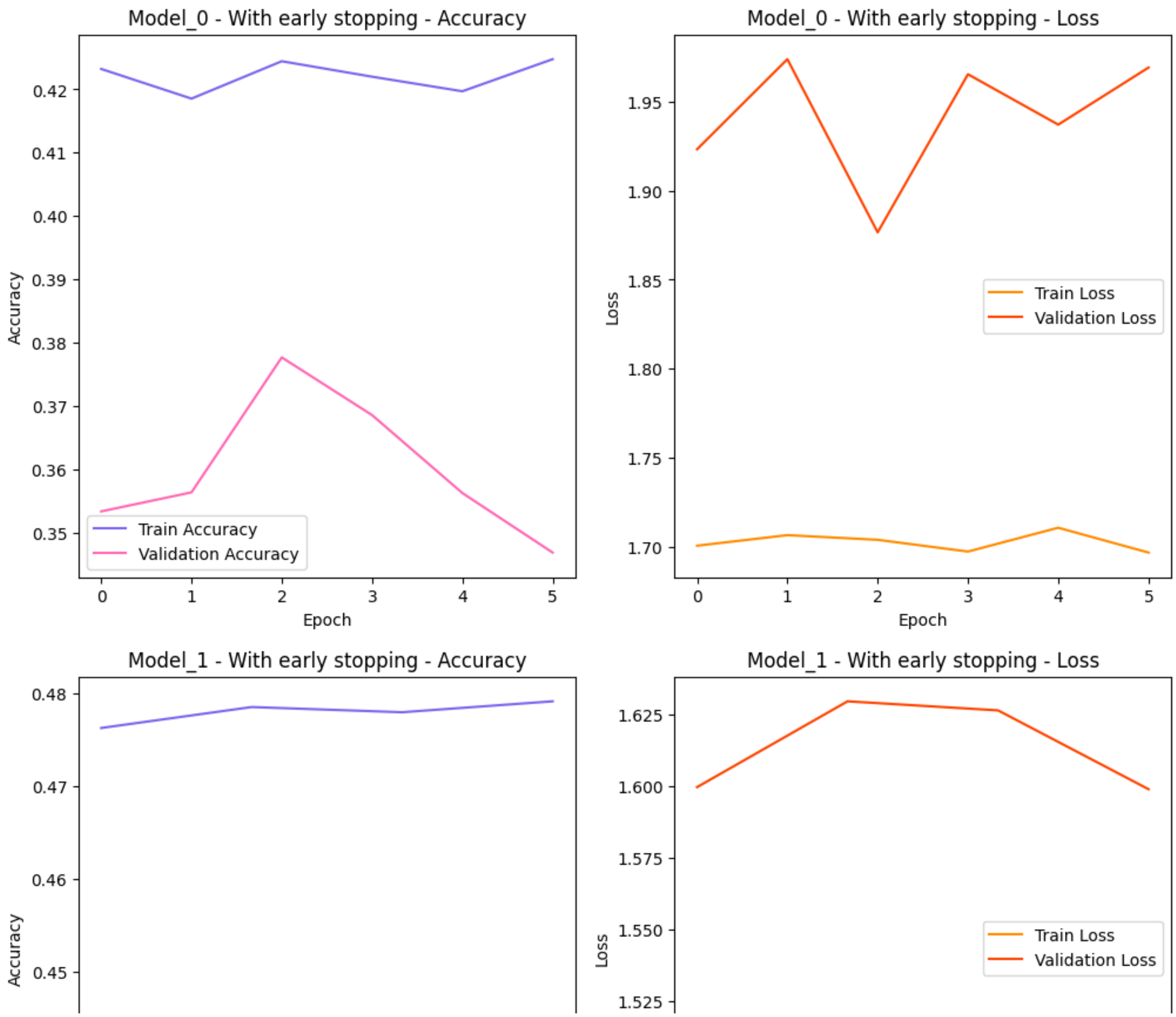
```
Epoch 3/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.5051 - accu
Epoch 4/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.5021 - accu
Epoch 5/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.5035 - accu
Epoch 6/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.5021 - accu
Epoch 7/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.4992 - accu
Epoch 8/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.4986 - accu
Epoch 9/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.4969 - accu
Epoch 10/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.4957 - accu
Epoch 11/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.4972 - accu
Epoch 12/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.4946 - accu
Epoch 13/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.4936 - accu
Epoch 14/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.4896 - accu
Epoch 15/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.4943 - accu
Epoch 16/50
1563/1563 [==============================] - 6s 4ms/step - loss: 1.4909 - accu
Epoch 17/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.4893 - accu
Epoch 18/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.4846 - accu
Epoch 19/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.4876 - accu
Epoch 20/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.4848 - accu
Epoch 21/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.4830 - accu
Epoch 22/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.4813 - accu
Epoch 23/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.4805 - accu
Epoch 24/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.4808 - accu
Epoch 25/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.4786 - accu
Epoch 26/50
1563/1563 [==============================] - 6s 4ms/step - loss: 1.4798 - accu
Epoch 27/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.4800 - accu
Epoch 28/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.4794 - accu
Epoch 29/50
```

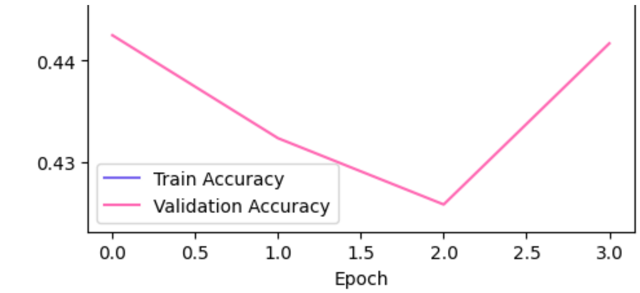
```
Epoch 29/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.4740 - accu
Epoch 30/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.4739 - accu
Epoch 31/50
1563/1563 [==============================] - 6s 4ms/step - loss: 1.4734 - accu
Epoch 32/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.4754 - accu
Epoch 33/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.4712 - accu
Epoch 34/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.4697 - accu
Epoch 35/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.4720 - accu
Epoch 36/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.4690 - accu
Epoch 37/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.4700 - accu
Epoch 38/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.4686 - accu
Epoch 39/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.4690 - accu
Epoch 40/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.4668 - accu
Epoch 41/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.4652 - accu
Epoch 42/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.4675 - accu
Epoch 43/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.4654 - accu
Epoch 44/50
1563/1563 [==============================] - 6s 4ms/step - loss: 1.4643 - accu
Epoch 45/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.4648 - accu
Epoch 46/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.4655 - accu
Epoch 47/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.4605 - accu
Epoch 48/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.4656 - accu
Epoch 49/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.4601 - accu
Epoch 50/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.4583 - accu
313/313 [==============================] - 1s 2ms/step - loss: 1.6163 - accura
Model_1 without early stopping - Test Loss: 1.6162830591201782, Test Accuracy
Training model_1 with early stopping...
Epoch 1/10
1563/1563 [==============================] - 6s 4ms/step - loss: 1.4590 - accu
Epoch 2/10
1563/1563 [==============================] - 5s 3ms/step - loss: 1.4604 - accu
Epoch 3/10
1563/1563 [==============================] - 6s 4ms/step - loss: 1.4580 - accu
```

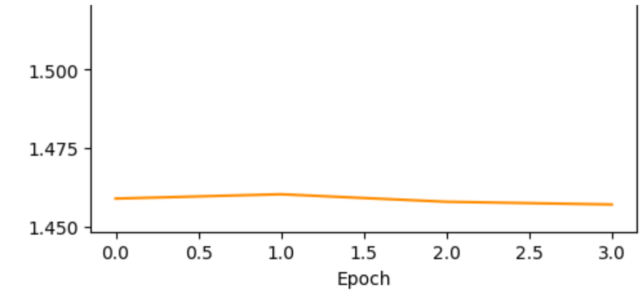
```
1563/1563 [==============================] - 6s 4ms/step - loss: 1.4580 - accu
Epoch 4/10
1563/1563 [==============================] - 5s 3ms/step - loss: 1.4572 - accu
313/313 [==============================] - 1s 2ms/step - loss: 1.5990 - accura
Model_1 with early stopping - Test Loss: 1.598970651626587, Test Accuracy: 0.4
Training model_2 without early stopping...
Epoch 1/50
1563/1563 [==============================] - 6s 4ms/step - loss: 2.1184 - accu
Epoch 2/50
1563/1563 [==============================] - 5s 3ms/step - loss: 2.0328 - accu
Epoch 3/50
1563/1563 [==============================] - 6s 4ms/step - loss: 1.9498 - accu
Epoch 4/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.9279 - accu
Epoch 5/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.9075 - accu
Epoch 6/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8931 - accu
Epoch 7/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8814 - accu
Epoch 8/50
1563/1563 [==============================] - 6s 4ms/step - loss: 1.8732 - accu
Epoch 9/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8634 - accu
Epoch 10/50
1563/1563 [==============================] - 6s 4ms/step - loss: 1.8612 - accu
Epoch 11/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8581 - accu
Epoch 12/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8529 - accu
Epoch 13/50
1563/1563 [==============================] - 6s 4ms/step - loss: 1.8512 - accu
Epoch 14/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8472 - accu
Epoch 15/50
1563/1563 [==============================] - 6s 4ms/step - loss: 1.8454 - accu
Epoch 16/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8417 - accu
Epoch 17/50
1563/1563 [==============================] - 6s 4ms/step - loss: 1.8415 - accu
Epoch 18/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8377 - accu
Epoch 19/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8371 - accu
Epoch 20/50
1563/1563 [==============================] - 6s 4ms/step - loss: 1.8349 - accu
Epoch 21/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8348 - accu
Epoch 22/50
1563/1563 [==============================] - 6s 4ms/step - loss: 1.8329 - accu
Epoch 23/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8292 - accu
Epoch 24/50
```

```
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8328 - acc
Epoch 25/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8292 - acc
Epoch 26/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8283 - acc
Epoch 27/50
1563/1563 [==============================] - 6s 4ms/step - loss: 1.8267 - acc
Epoch 28/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8259 - acc
Epoch 29/50
1563/1563 [==============================] - 6s 4ms/step - loss: 1.8264 - acc
Epoch 30/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8243 - acc
Epoch 31/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8233 - acc
Epoch 32/50
1563/1563 [==============================] - 6s 4ms/step - loss: 1.8258 - acc
Epoch 33/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8240 - acc
Epoch 34/50
1563/1563 [==============================] - 6s 4ms/step - loss: 1.8211 - acc
Epoch 35/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8223 - acc
Epoch 36/50
1563/1563 [==============================] - 6s 4ms/step - loss: 1.8207 - acc
Epoch 37/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8213 - acc
Epoch 38/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8212 - acc
Epoch 39/50
1563/1563 [==============================] - 6s 4ms/step - loss: 1.8166 - acc
Epoch 40/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8193 - acc
Epoch 41/50
1563/1563 [==============================] - 6s 4ms/step - loss: 1.8173 - acc
Epoch 42/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8154 - acc
Epoch 43/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8162 - acc
Epoch 44/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8149 - acc
Epoch 45/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8148 - acc
Epoch 46/50
1563/1563 [==============================] - 7s 4ms/step - loss: 1.8169 - acc
Epoch 47/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8137 - acc
Epoch 48/50
1563/1563 [==============================] - 6s 4ms/step - loss: 1.8131 - acc
Epoch 49/50
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8115 - acc
Epoch 50/50
```

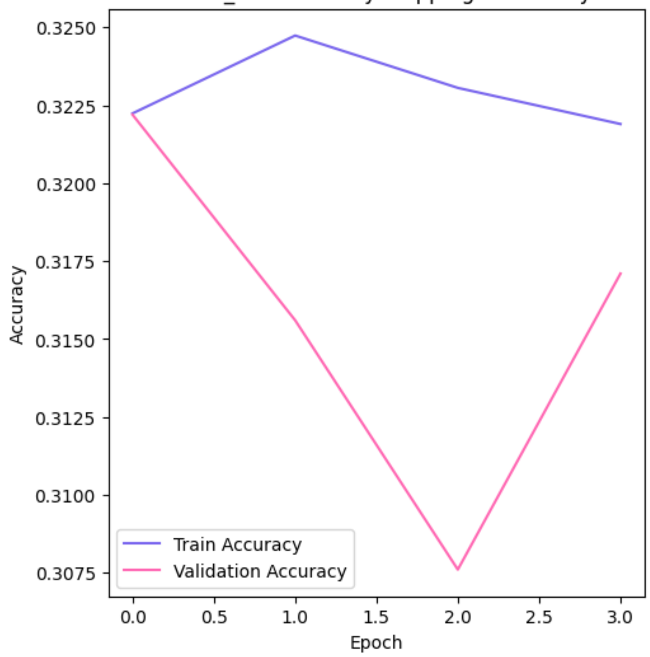
```
1563/1563 [==============================] - 6s 4ms/step - loss: 1.8130 - accu
313/313 [==============================] - 1s 2ms/step - loss: 1.8366 - accura
Model_2 without early stopping - Test Loss: 1.8366405963897705, Test Accuracy
Training model_2 with early stopping...
Epoch 1/10
1563/1563 [==============================] - 7s 4ms/step - loss: 1.8139 - accu
Epoch 2/10
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8120 - accu
Epoch 3/10
1563/1563 [==============================] - 6s 4ms/step - loss: 1.8102 - accu
Epoch 4/10
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8127 - accu
313/313 [==============================] - 1s 3ms/step - loss: 1.8174 - accura
Model_2 with early stopping - Test Loss: 1.817396640777588, Test Accuracy: 0.:

============================== Print Statements ==============================
Model_0 without early stopping - Test Loss: 2.009751081466675, Test Accuracy:
Model_0 with early stopping - Test Loss: 1.9690543413162231, Test Accuracy: 0
Model_1 without early stopping - Test Loss: 1.6162830591201782, Test Accuracy
Model_1 with early stopping - Test Loss: 1.598970651626587, Test Accuracy: 0.
Model_2 without early stopping - Test Loss: 1.8366405963897705, Test Accuracy
Model_2 with early stopping - Test Loss: 1.817396640777588, Test Accuracy: 0.:
```



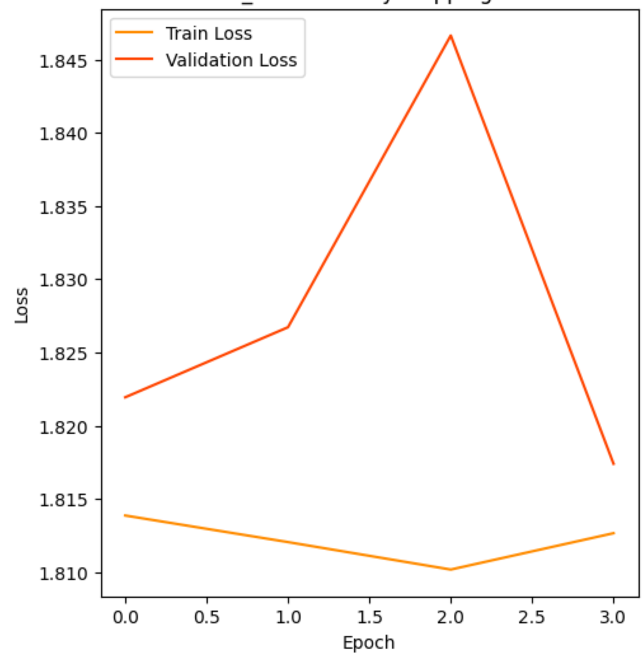




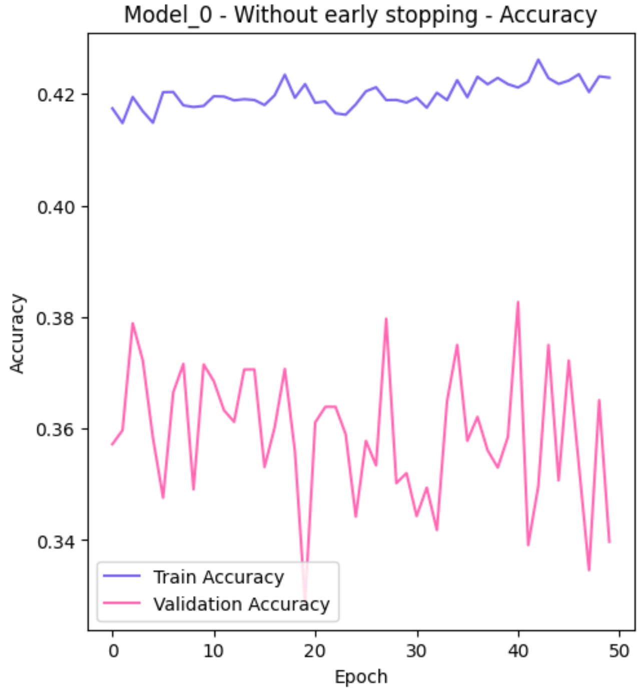
Model_2 - With early stopping - Accuracy



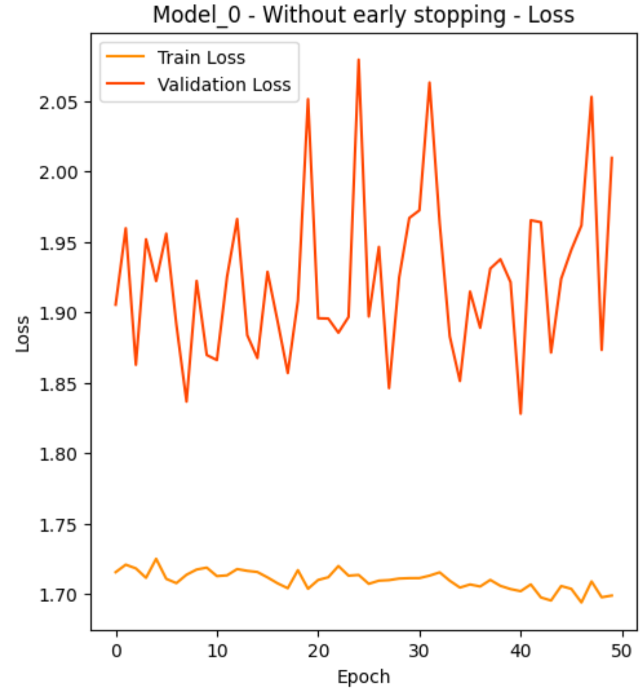
Model_2 - With early stopping - Loss



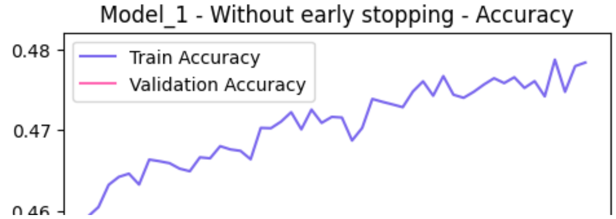
Model_0 - Without early stopping - Accuracy



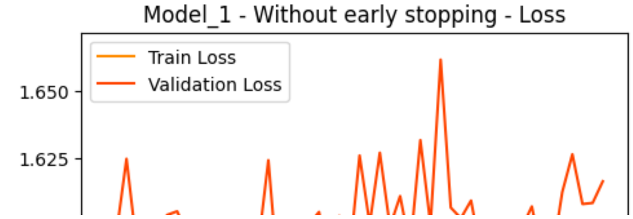
Model_0 - Without early stopping - Loss

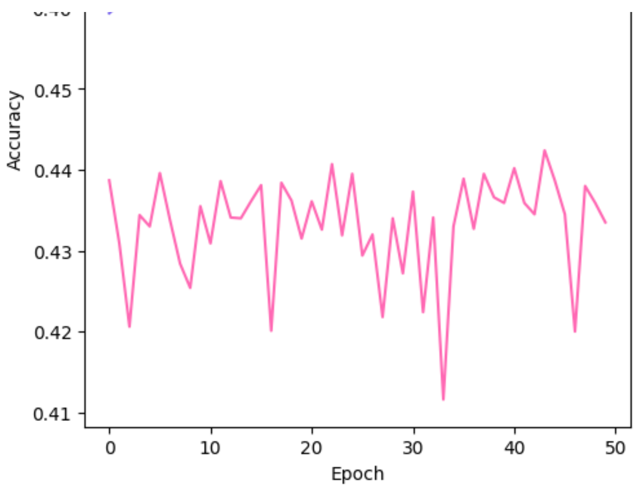


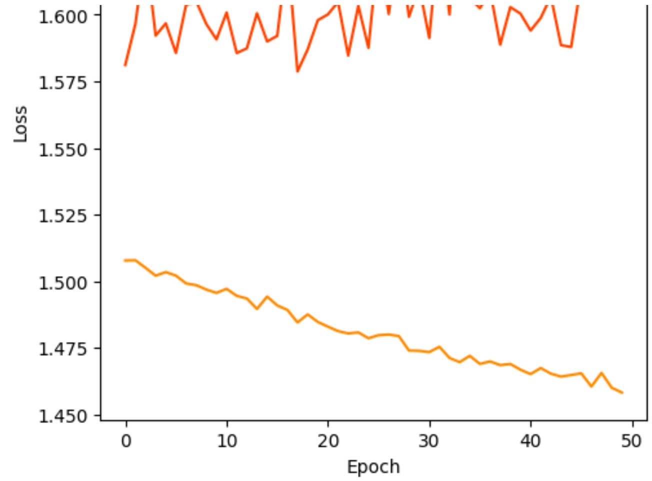
Model_1 - Without early stopping - Accuracy

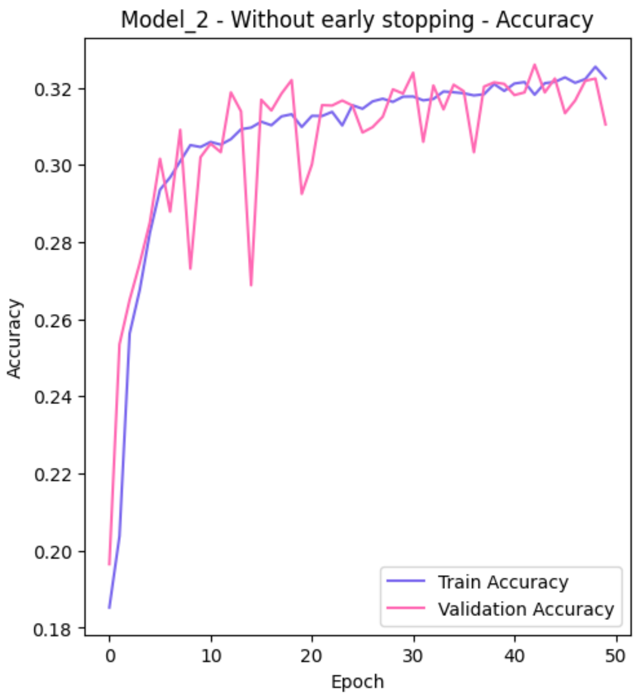


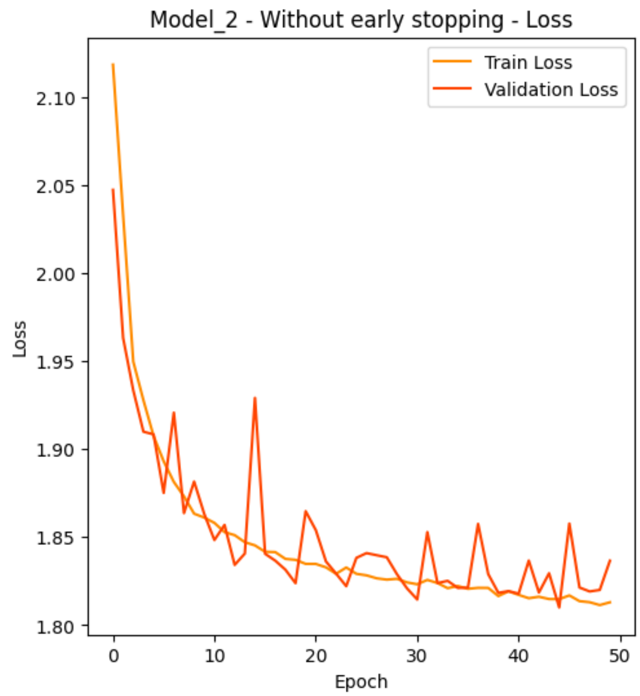
Model_1 - Without early stopping - Loss




Accuracy
Epoch


Loss
Epoch


Model_2 - Without early stopping - Accuracy
Accuracy
Epoch
Train Accuracy
Validation Accuracy


Model_2 - Without early stopping - Loss
Loss
Epoch
Train Loss
Validation Loss

Discussion

The obtained results are:

Model 0 without early stopping - Test Loss: 2.010, Test Accuracy: 0.3397

Model 0 with early stopping - Test Loss: 1.969, Test Accuracy: 0.3478

Model 1 without early stopping - Test Loss: 1.616, Test Accuracy: 0.4335
Model_1 with early stopping - Test Loss: 1.598, Test Accuracy: 0.4417

Model_2 without early stopping - Test Loss: 1.836, Test Accuracy: 0.3104 Model_2 with early stopping - Test Loss: 1.8173 Test Accuracy: 0.3171

Both the validation loss and accuracy are very noisy. This makes it hard to properly implement an early stopping mechanism and this renders the training with early stopping basically useless. If one could smooth out this validation stochastisty, early stopping might become viable.

The model with no hidden layer seems unable to learn, it's validation accuracy wildly varies, while its training accuracy increases very slowly (1% in 20 epochs)

The model with a single hidden layer also has a very hard time learning, the validation fluctuations only being a bit smaller. Its training increase is double that of the 0 layer model (2% in 20 epochs) and it also scored significantly better on test accuracy (fluctuating around 43% instead of 36% val accuracy).

The model with two hidden layers is most interesting. The validation curves actually seem to follow the training curves, instead of being seperated from them. Initially it starts at a lot lower validation accuracy but it learns significantly more (20% increase in 20 epochs!). The fluctuations in the validation curves are still pretty significant though. The single-layer perceptron outperform the double-layer one significantly for small epochs (43% vs 31% validation accuracy) We expect that the two layer perceptron can eventually outperform the single-layer perceptron, given enough epochs.

We will test this below.

```
models_list = [model_2]
histories_without_es = []  # store history of each model without early stopping
histories_with_es = []     # store history of each model with early stopping
print_statements = []      # store print to print at the end

# train each model and plot
for i, model in enumerate(models_list):

    # train the model without early stopping [callback argument]
    print(f"Training model_{i} without early stopping...")
    history_without_es = model.fit(train_images, train_labels, epochs=150,
                                   validation_data=(test_images, test_labels))
    # evaluate and store
    histories_without_es.append(history_without_es)
    test_loss, test_acc = model.evaluate(test_images, test_labels)
    print_statement = f"Model_{i} without early stopping - Test Loss: {test_loss},
```

```
        print_statements.append(print_statement)
        print(print_statement)


    # print all statements again at the end
    print("\n" + "="*30 + " Print Statements " + "="*30)
    for statement in print_statements:
        print(statement)

    # plot learning curves
    def plot_learning_curves(histories, title):
        for i, history in enumerate(histories):
            plt.figure(figsize=(12, 6))

            plt.subplot(1, 2, 1)
            plt.plot(history.history['accuracy'], label='Train Accuracy', color = 'Medi
            plt.plot(history.history['val_accuracy'], label='Validation Accuracy', colo
            plt.title(f'Model_{i} - {title} - Accuracy')
            plt.xlabel('Epoch')
            plt.ylabel('Accuracy')
            plt.legend()

            plt.subplot(1, 2, 2)
            plt.plot(history.history['loss'], label='Train Loss', color = 'DarkOrange')
            plt.plot(history.history['val_loss'], label='Validation Loss', color = 'Ora
            plt.title(f'Model_{i} - {title} - Loss')
            plt.xlabel('Epoch')
            plt.ylabel('Loss')
            plt.legend()

            plt.show()

    plot_learning_curves(histories_with_es, 'With early stopping')
    plot_learning_curves(histories_without_es, 'Without early stopping')
```

```
Training model_0 without early stopping...
Epoch 1/150
1563/1563 [==============================] - 6s 4ms/step - loss: 1.8095 - accu
Epoch 2/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8100 - accu
Epoch 3/150
1563/1563 [==============================] - 6s 4ms/step - loss: 1.8086 - accu
Epoch 4/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8098 - accu
Epoch 5/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8088 - accu
Epoch 6/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8055 - accu
Epoch 7/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8058 - accu
Epoch 8/150
```

```
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8088 - accu
Epoch 9/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8065 - accu
Epoch 10/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8106 - accu
Epoch 11/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8101 - accu
Epoch 12/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8050 - accu
Epoch 13/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8070 - accu
Epoch 14/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8047 - accu
Epoch 15/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8065 - accu
Epoch 16/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8049 - accu
Epoch 17/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8030 - accu
Epoch 18/150
1563/1563 [==============================] - 6s 4ms/step - loss: 1.8040 - accu
Epoch 19/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8045 - accu
Epoch 20/150
1563/1563 [==============================] - 6s 4ms/step - loss: 1.8054 - accu
Epoch 21/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8040 - accu
Epoch 22/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8018 - accu
Epoch 23/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8001 - accu
Epoch 24/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8015 - accu
Epoch 25/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7991 - accu
Epoch 26/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8006 - accu
Epoch 27/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7989 - accu
Epoch 28/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7976 - accu
Epoch 29/150
1563/1563 [==============================] - 6s 4ms/step - loss: 1.7972 - accu
Epoch 30/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7959 - accu
Epoch 31/150
1563/1563 [==============================] - 6s 4ms/step - loss: 1.7950 - accu
Epoch 32/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7977 - accu
Epoch 33/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7967 - accu
Epoch 34/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7949 - accu
```

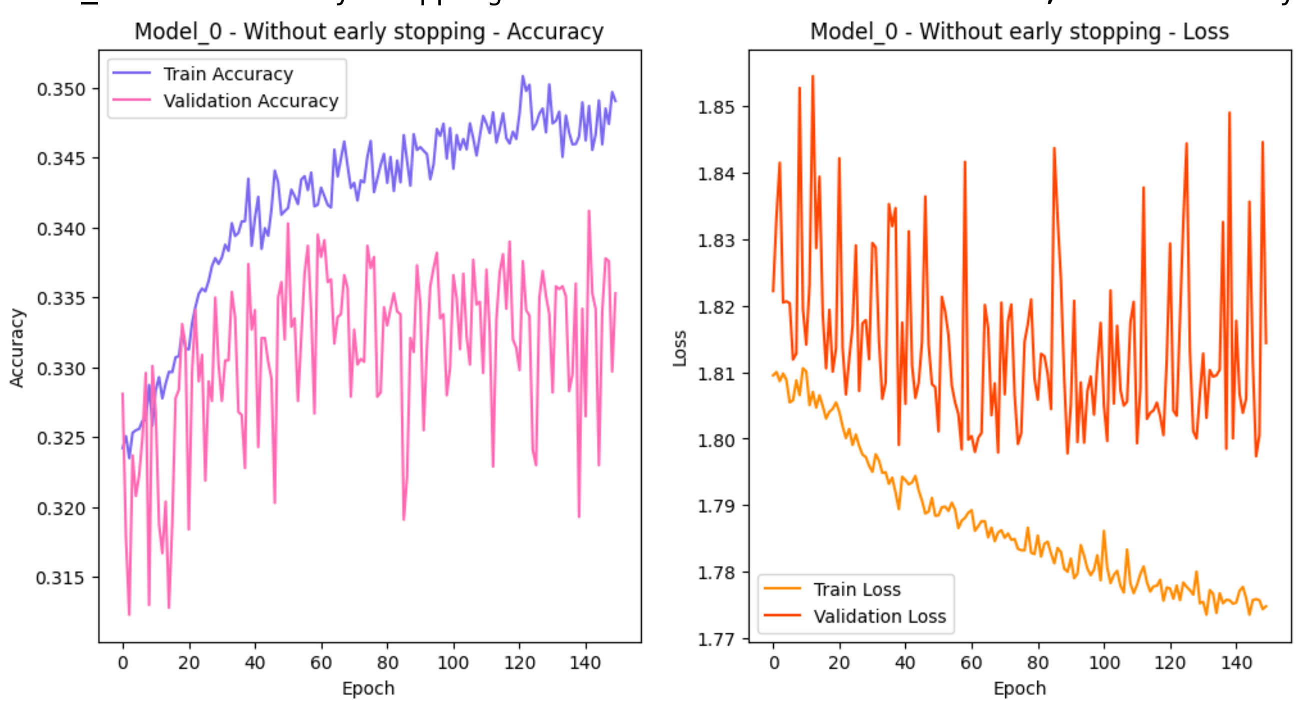
```
Epoch 35/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7949 - accu
Epoch 36/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7932 - accu
Epoch 37/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7941 - accu
Epoch 38/150
1563/1563 [==============================] - 6s 4ms/step - loss: 1.7919 - accu
Epoch 39/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7894 - accu
Epoch 40/150
1563/1563 [==============================] - 6s 4ms/step - loss: 1.7943 - accu
Epoch 41/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7938 - accu
Epoch 42/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7931 - accu
Epoch 43/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7934 - accu
Epoch 44/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7944 - accu
Epoch 45/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7922 - accu
Epoch 46/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7907 - accu
Epoch 47/150
1563/1563 [==============================] - 5s 4ms/step - loss: 1.7887 - accu
Epoch 48/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7890 - accu
Epoch 49/150
1563/1563 [==============================] - 6s 4ms/step - loss: 1.7911 - accu
Epoch 50/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7884 - accu
Epoch 51/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7885 - accu
Epoch 52/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7897 - accu
Epoch 53/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7898 - accu
Epoch 54/150
1563/1563 [==============================] - 6s 4ms/step - loss: 1.7891 - accu
Epoch 55/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7904 - accu
Epoch 56/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7893 - accu
Epoch 57/150
1563/1563 [==============================] - 6s 4ms/step - loss: 1.7865 - accu
Epoch 58/150
1563/1563 [==============================] - 6s 4ms/step - loss: 1.7877 - accu
Epoch 59/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7881 - accu
Epoch 60/150
1563/1563 [==============================] - 6s 4ms/step - loss: 1.7889 - accu
```

```
Epoch 61/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7892 - acc
Epoch 62/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7861 - acc
Epoch 63/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7868 - acc
Epoch 64/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7876 - acc
Epoch 65/150
1563/1563 [==============================] - 5s 4ms/step - loss: 1.7876 - acc
Epoch 66/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7851 - acc
Epoch 67/150
1563/1563 [==============================] - 5s 4ms/step - loss: 1.7866 - acc
Epoch 68/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7846 - acc
Epoch 69/150
1563/1563 [==============================] - 5s 4ms/step - loss: 1.7859 - acc
Epoch 70/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7862 - acc
Epoch 71/150
1563/1563 [==============================] - 6s 4ms/step - loss: 1.7851 - acc
Epoch 72/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7857 - acc
Epoch 73/150
1563/1563 [==============================] - 6s 4ms/step - loss: 1.7848 - acc
Epoch 74/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7849 - acc
Epoch 75/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7834 - acc
Epoch 76/150
1563/1563 [==============================] - 5s 4ms/step - loss: 1.7832 - acc
Epoch 77/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7832 - acc
Epoch 78/150
1563/1563 [==============================] - 6s 4ms/step - loss: 1.7866 - acc
Epoch 79/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7828 - acc
Epoch 80/150
1563/1563 [==============================] - 6s 4ms/step - loss: 1.7826 - acc
Epoch 81/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7854 - acc
Epoch 82/150
1563/1563 [==============================] - 6s 4ms/step - loss: 1.7823 - acc
Epoch 83/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7842 - acc
Epoch 84/150
1563/1563 [==============================] - 6s 4ms/step - loss: 1.7845 - acc
Epoch 85/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7827 - acc
Epoch 86/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7813 - acc
```

```
Epoch 87/150
1563/1563 [==============================] - 6s 4ms/step - loss: 1.7835 - accu
Epoch 88/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7830 - accu
Epoch 89/150
1563/1563 [==============================] - 6s 4ms/step - loss: 1.7804 - accu
Epoch 90/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7800 - accu
Epoch 91/150
1563/1563 [==============================] - 6s 4ms/step - loss: 1.7820 - accu
Epoch 92/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7790 - accu
Epoch 93/150
1563/1563 [==============================] - 6s 4ms/step - loss: 1.7797 - accu
Epoch 94/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7840 - accu
Epoch 95/150
1563/1563 [==============================] - 6s 4ms/step - loss: 1.7824 - accu
Epoch 96/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7805 - accu
Epoch 97/150
1563/1563 [==============================] - 6s 4ms/step - loss: 1.7795 - accu
Epoch 98/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7803 - accu
Epoch 99/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7825 - accu
Epoch 100/150
1563/1563 [==============================] - 6s 4ms/step - loss: 1.7787 - accu
Epoch 101/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7861 - accu
Epoch 102/150
1563/1563 [==============================] - 6s 4ms/step - loss: 1.7806 - accu
Epoch 103/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7783 - accu
Epoch 104/150
1563/1563 [==============================] - 6s 4ms/step - loss: 1.7795 - accu
Epoch 105/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7801 - accu
Epoch 106/150
1563/1563 [==============================] - 6s 4ms/step - loss: 1.7779 - accu
Epoch 107/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7768 - accu
Epoch 108/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7833 - accu
Epoch 109/150
1563/1563 [==============================] - 6s 4ms/step - loss: 1.7782 - accu
Epoch 110/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7768 - accu
Epoch 111/150
1563/1563 [==============================] - 6s 4ms/step - loss: 1.7780 - accu
Epoch 112/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7796 - accu
Epoch 113/150
```

```
Epoch 113/150
1563/1563 [==============================] - 6s 4ms/step - loss: 1.7808 - accu
Epoch 114/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7784 - accu
Epoch 115/150
1563/1563 [==============================] - 6s 4ms/step - loss: 1.7770 - accu
Epoch 116/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7778 - accu
Epoch 117/150
1563/1563 [==============================] - 6s 4ms/step - loss: 1.7779 - accu
Epoch 118/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7788 - accu
Epoch 119/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7756 - accu
Epoch 120/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7776 - accu
Epoch 121/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7775 - accu
Epoch 122/150
1563/1563 [==============================] - 6s 4ms/step - loss: 1.7759 - accu
Epoch 123/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7779 - accu
Epoch 124/150
1563/1563 [==============================] - 6s 4ms/step - loss: 1.7757 - accu
Epoch 125/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7784 - accu
Epoch 126/150
1563/1563 [==============================] - 6s 4ms/step - loss: 1.7777 - accu
Epoch 127/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7772 - accu
Epoch 128/150
1563/1563 [==============================] - 6s 4ms/step - loss: 1.7765 - accu
Epoch 129/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7800 - accu
Epoch 130/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7752 - accu
Epoch 131/150
1563/1563 [==============================] - 5s 4ms/step - loss: 1.7755 - accu
Epoch 132/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7735 - accu
Epoch 133/150
1563/1563 [==============================] - 6s 4ms/step - loss: 1.7772 - accu
Epoch 134/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7767 - accu
Epoch 135/150
1563/1563 [==============================] - 5s 4ms/step - loss: 1.7738 - accu
Epoch 136/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7767 - accu
Epoch 137/150
1563/1563 [==============================] - 6s 4ms/step - loss: 1.7752 - accu
Epoch 138/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7758 - accu
Epoch 139/150
```

```
1563/1563 [==============================] - 7s 4ms/step - loss: 1.7757 - accu
Epoch 140/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7752 - accu
Epoch 141/150
1563/1563 [==============================] - 6s 4ms/step - loss: 1.7753 - accu
Epoch 142/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7771 - accu
Epoch 143/150
1563/1563 [==============================] - 6s 4ms/step - loss: 1.7777 - accu
Epoch 144/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7760 - accu
Epoch 145/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7735 - accu
Epoch 146/150
1563/1563 [==============================] - 6s 4ms/step - loss: 1.7757 - accu
Epoch 147/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7759 - accu
Epoch 148/150
1563/1563 [==============================] - 6s 4ms/step - loss: 1.7757 - accu
Epoch 149/150
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7743 - accu
Epoch 150/150
1563/1563 [==============================] - 6s 4ms/step - loss: 1.7748 - accu
313/313 [==============================] - 1s 2ms/step - loss: 1.8144 - accura
Model_0 without early stopping - Test Loss: 1.8143916130065918, Test Accuracy

============================== Print Statements ==============================
Model_0 without early stopping - Test Loss: 1.8143916130065918, Test Accuracy
```



Note: the plot says model 0, but its actually model 2. Interestingly, learning accuracy only improved by 2%. The validation fluctuations are still very severe. This means the single-layer MLP has the best complexity of these models to fit the data correctly.

Exercise 2 - Large Single Layer MLP

Reuse the code from part 1 to build and run a MLP with one hidden layer as big a you can. Compare the performance of your design with the results appearing in Table 1 of [https://arxiv.org/pdf/1611.03530.pdf] for a MLP of 512 units in a single hidden layer. Report the best result found for a maximum of 1000 epochs or 2 hrs CPU running time. The best accuracy amongst all teams will be awarded extra points.

Load model or create new model

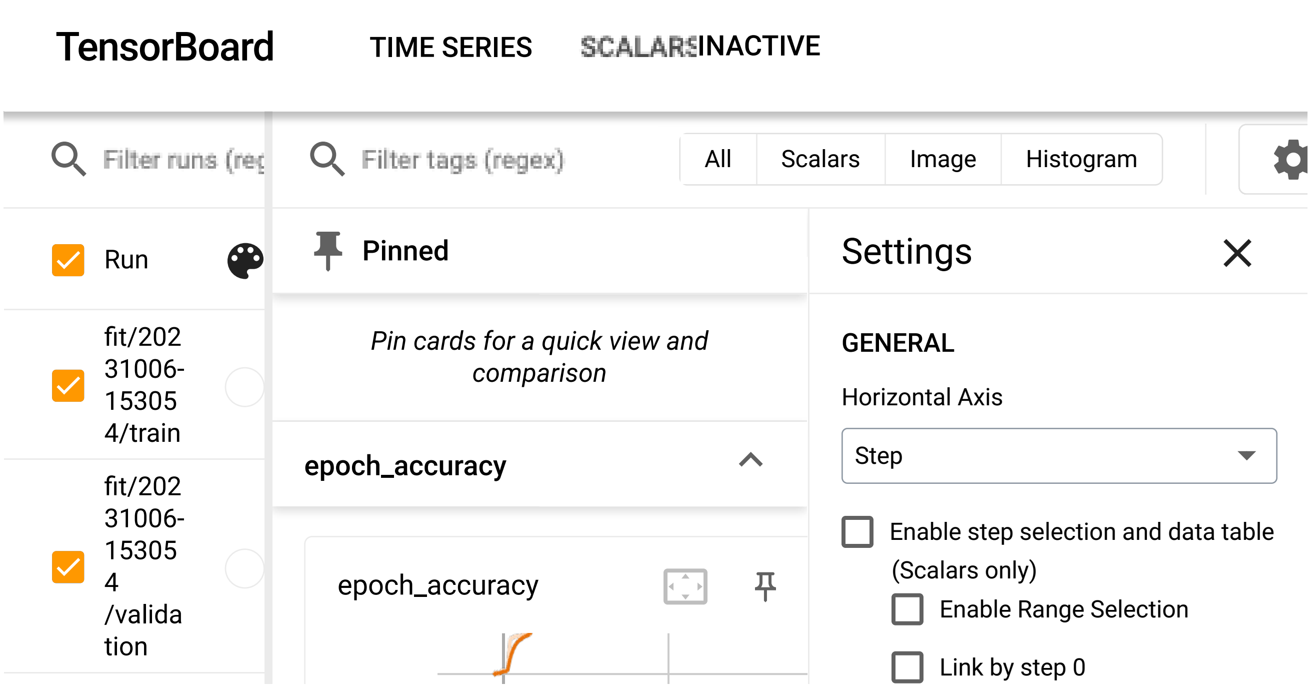
```
#hyperparameter tuning
load_model = False
units = 2048
learning_rate = 0.0005

if load_model:
    # load the saved best model if there is one
    try:
        model_large_hidden = tf.keras.models.load_model('best_model.h5')
        print("No saved model found, create a new model.")
    except OSError:
        pass
else:
        # otherwise train a new one
        model_large_hidden = models.Sequential([
        layers.Flatten(),
        layers.Dense(units = units, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])

        opt = tf.keras.optimizers.legacy.Adam(learning_rate= learning_rate)
        model_large_hidden.compile(optimizer=opt,
                                   loss='sparse_categorical_crossentropy',
```

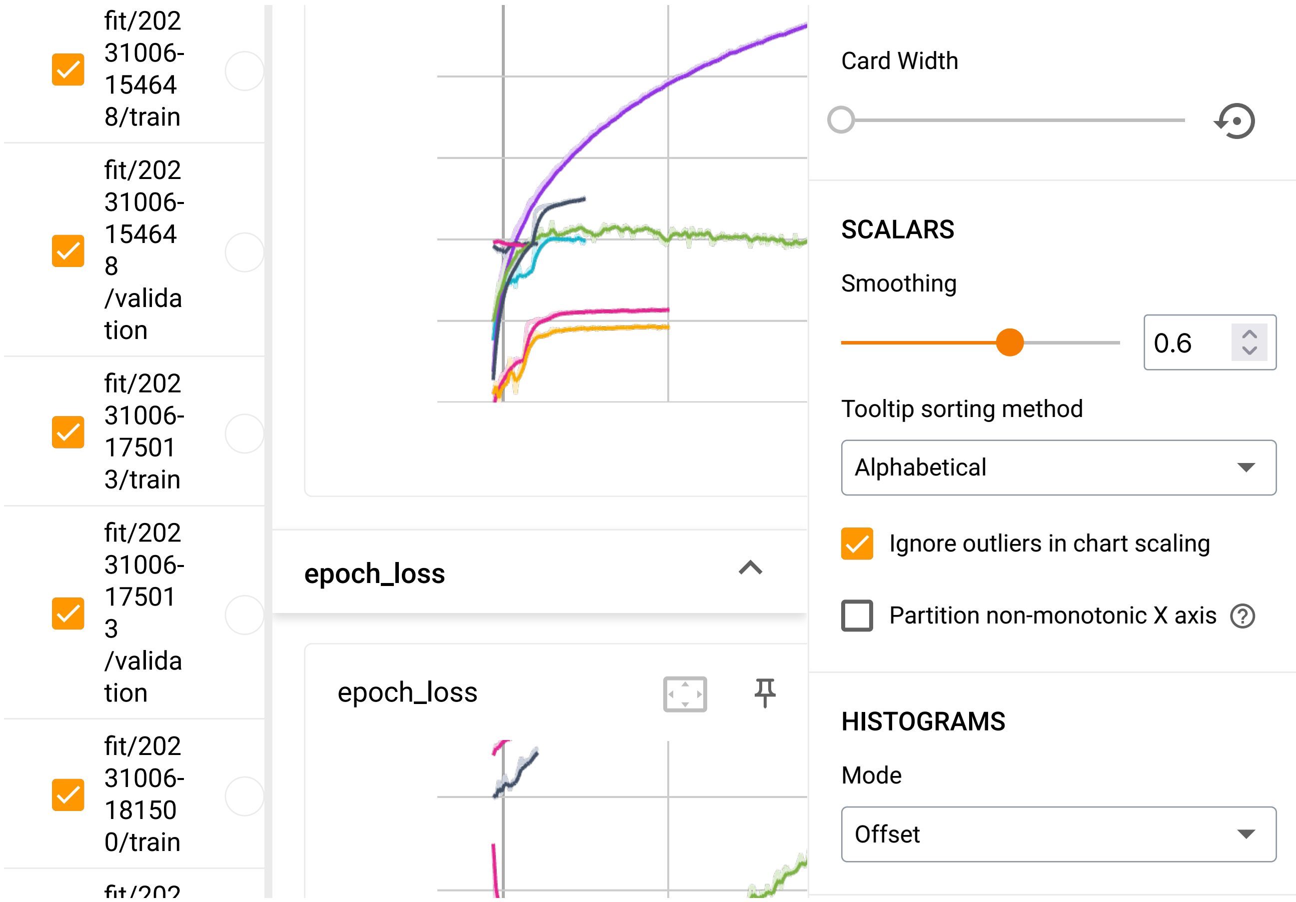
```
                                        metrics=['accuracy'])
```

Import callbacks

```
# use checkpoint callback to save the best model
checkpoint_filepath = 'best_model.h5'  # file path for best model
checkpoint = tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_filepath,
    save_weights_only=False,
    monitor='val_accuracy',
    mode='max',                # save max val accuracy
    save_best_only=True,       # only save the best model
    verbose=1)                 # monitor progress

# use early stopping
early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=

# reduce learning rate on plateaus
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_accuracy', factor=0.5

# monitor during training callback
log_dir = "./logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard = tf.keras.callbacks.TensorBoard(log_dir=log_dir)
%load_ext tensorboard
%tensorboard --logdir ./logs/
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
Reusing TensorBoard on port 6006 (pid 5502), started 2:49:52 ago. (Use '!kill
5502' to kill it.)
```





Train the large model

```
# train the model
start_time = time.time()
history_large_hidden = model_large_hidden.fit(train_images, train_labels, epochs=15
                                              validation_data=(test_images, test_la
                                              callbacks=[checkpoint, early_stopping
                                              verbose=1)  # monitor progress
end_time = time.time()
test_loss, test_acc = model_large_hidden.evaluate(test_images, test_labels, verbose
print(f"Training Time: {end_time - start_time} seconds")
print(f"Test Loss: {test_loss}, Test Accuracy: {test_acc}")

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.plot(history_large_hidden.history['accuracy'], label='Train Accuracy', color =
plt.plot(history_large_hidden.history['val_accuracy'], label='Validation Accuracy',
plt.title(f'Large Single Layer Model - Loss')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
```

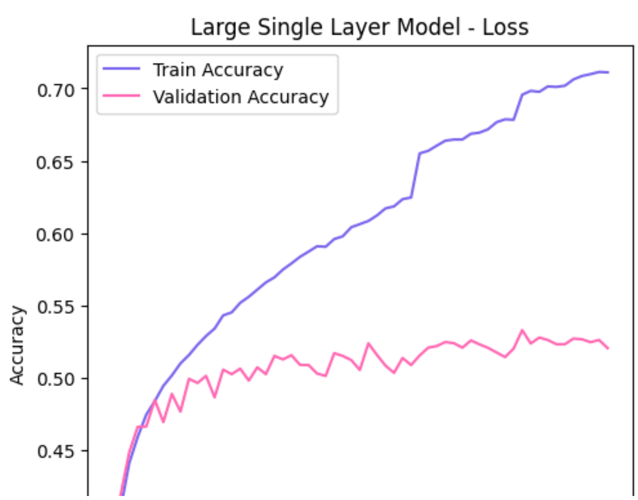
```
plt.plot(history_large_hidden.history['loss'], label='Train Loss', color = 'DarkOra
plt.plot(history_large_hidden.history['val_loss'], label='Validation Loss', color =
plt.title(f'Large Single Layer Model - Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.show()

    Epoch 1/150
    1553/1563 [============================>.] - ETA: 0s - loss: 1.8715 - accurac
    Epoch 1: val_accuracy improved from -inf to 0.38280, saving model to best_mod
    1563/1563 [==============================] - 8s 5ms/step - loss: 1.8701 - acc
    Epoch 2/150
    1551/1563 [============================>.] - ETA: 0s - loss: 1.6611 - accurac
    Epoch 2: val_accuracy improved from 0.38280 to 0.41890, saving model to best_
    1563/1563 [==============================] - 6s 4ms/step - loss: 1.6608 - acc
    Epoch 3/150
    1560/1563 [============================>.] - ETA: 0s - loss: 1.5784 - accurac
    Epoch 3: val_accuracy improved from 0.41890 to 0.44820, saving model to best_
    1563/1563 [==============================] - 7s 5ms/step - loss: 1.5785 - acc
    Epoch 4/150
    1560/1563 [============================>.] - ETA: 0s - loss: 1.5267 - accurac
    Epoch 4: val_accuracy improved from 0.44820 to 0.46610, saving model to best_
    1563/1563 [==============================] - 6s 4ms/step - loss: 1.5266 - acc
    Epoch 5/150
    1551/1563 [============================>.] - ETA: 0s - loss: 1.4865 - accurac
    Epoch 5: val_accuracy improved from 0.46610 to 0.46630, saving model to best_
    1563/1563 [==============================] - 7s 5ms/step - loss: 1.4859 - acc
    Epoch 6/150
    1555/1563 [============================>.] - ETA: 0s - loss: 1.4538 - accurac
    Epoch 6: val_accuracy improved from 0.46630 to 0.48460, saving model to best_
    1563/1563 [==============================] - 6s 4ms/step - loss: 1.4542 - acc
    Epoch 7/150
    1561/1563 [============================>.] - ETA: 0s - loss: 1.4291 - accurac
    Epoch 7: val_accuracy did not improve from 0.48460
    1563/1563 [==============================] - 6s 4ms/step - loss: 1.4293 - acc
    Epoch 8/150
    1561/1563 [============================>.] - ETA: 0s - loss: 1.4067 - accurac
    Epoch 8: val_accuracy improved from 0.48460 to 0.48890, saving model to best_
    1563/1563 [==============================] - 6s 4ms/step - loss: 1.4067 - acc
    Epoch 9/150
    1555/1563 [============================>.] - ETA: 0s - loss: 1.3834 - accurac
    Epoch 9: val_accuracy did not improve from 0.48890
    1563/1563 [==============================] - 7s 4ms/step - loss: 1.3833 - acc
    Epoch 10/150
    1558/1563 [============================>.] - ETA: 0s - loss: 1.3636 - accurac
    Epoch 10: val_accuracy improved from 0.48890 to 0.49930, saving model to best
    1563/1563 [==============================] - 6s 4ms/step - loss: 1.3636 - acc
    Epoch 11/150
    1563/1563 [==============================] - ETA: 0s - loss: 1.3483 - accurac
    Epoch 11: val_accuracy did not improve from 0.49930
```

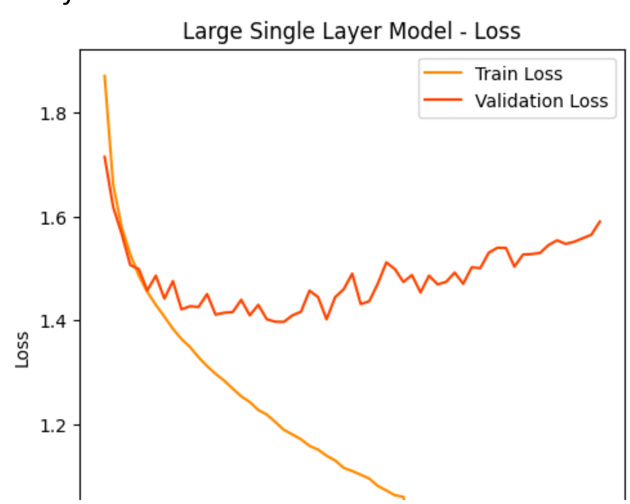
```
1563/1563 [==============================] - 6s 4ms/step - loss: 1.3483 - accu
Epoch 12/150
1560/1563 [============================>.] - ETA: 0s - loss: 1.3284 - accurac
Epoch 12: val_accuracy improved from 0.49930 to 0.50130, saving model to best_
1563/1563 [==============================] - 6s 4ms/step - loss: 1.3285 - accu
Epoch 13/150
1558/1563 [============================>.] - ETA: 0s - loss: 1.3112 - accurac
Epoch 13: val_accuracy did not improve from 0.50130
1563/1563 [==============================] - 7s 4ms/step - loss: 1.3115 - accu
Epoch 14/150
1563/1563 [==============================] - ETA: 0s - loss: 1.2964 - accurac
Epoch 14: val_accuracy improved from 0.50130 to 0.50560, saving model to best_
1563/1563 [==============================] - 6s 4ms/step - loss: 1.2964 - accu
Epoch 15/150
1562/1563 [============================>.] - ETA: 0s - loss: 1.2836 - accurac
Epoch 15: val_accuracy did not improve from 0.50560
1563/1563 [==============================] - 7s 4ms/step - loss: 1.2836 - accu
Epoch 16/150
1549/1563 [============================>.] - ETA: 0s - loss: 1.2683 - accurac
Epoch 16: val_accuracy improved from 0.50560 to 0.50630, saving model to best_
1563/1563 [==============================] - 6s 4ms/step - loss: 1.2681 - accu
Epoch 17/150
1562/1563 [============================>.] - ETA: 0s - loss: 1.2533 - accurac
Epoch 17: val_accuracy did not improve from 0.50630
1563/1563 [==============================] - 7s 4ms/step - loss: 1.2533 - accu
Epoch 18/150
1555/1563 [============================>.] - ETA: 0s - loss: 1.2426 - accurac
Epoch 18: val_accuracy improved from 0.50630 to 0.50720, saving model to best_
1563/1563 [==============================] - 6s 4ms/step - loss: 1.2424 - accu
Epoch 19/150
1554/1563 [============================>.] - ETA: 0s - loss: 1.2269 - accurac
Epoch 19: val_accuracy did not improve from 0.50720
1563/1563 [==============================] - 7s 4ms/step - loss: 1.2269 - accu
Epoch 20/150
1555/1563 [============================>.] - ETA: 0s - loss: 1.2180 - accurac
Epoch 20: val_accuracy improved from 0.50720 to 0.51510, saving model to best_
1563/1563 [==============================] - 6s 4ms/step - loss: 1.2180 - accu
Epoch 21/150
1558/1563 [============================>.] - ETA: 0s - loss: 1.2041 - accurac
Epoch 21: val_accuracy did not improve from 0.51510
1563/1563 [==============================] - 6s 4ms/step - loss: 1.2038 - accu
Epoch 22/150
1562/1563 [============================>.] - ETA: 0s - loss: 1.1887 - accurac
Epoch 22: val_accuracy improved from 0.51510 to 0.51560, saving model to best_
1563/1563 [==============================] - 7s 4ms/step - loss: 1.1886 - accu
Epoch 23/150
1561/1563 [============================>.] - ETA: 0s - loss: 1.1798 - accurac
Epoch 23: val_accuracy did not improve from 0.51560
1563/1563 [==============================] - 6s 4ms/step - loss: 1.1798 - accu
Epoch 24/150
1559/1563 [============================>.] - ETA: 0s - loss: 1.1705 - accurac
Epoch 24: val_accuracy did not improve from 0.51560
1563/1563 [==============================] - 6s 4ms/step - loss: 1.1706 - accu
```

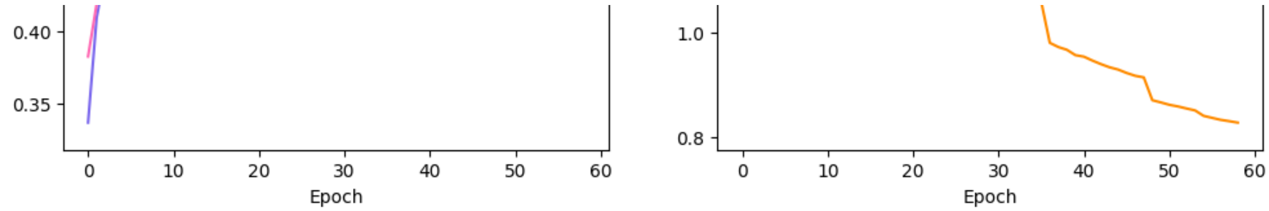
```
1563/1563 [==============================] - 6s 4ms/step - loss: 1.1706 - accu
Epoch 25/150
1554/1563 [============================>.] - ETA: 0s - loss: 1.1582 - accuracy
Epoch 25: val_accuracy did not improve from 0.51560
1563/1563 [==============================] - 6s 4ms/step - loss: 1.1578 - accu
Epoch 26/150
1554/1563 [============================>.] - ETA: 0s - loss: 1.1507 - accuracy
Epoch 26: val_accuracy did not improve from 0.51560
1563/1563 [==============================] - 7s 4ms/step - loss: 1.1507 - accu
Epoch 27/150
1559/1563 [============================>.] - ETA: 0s - loss: 1.1386 - accuracy
Epoch 27: val_accuracy improved from 0.51560 to 0.51700, saving model to best_
1563/1563 [==============================] - 6s 4ms/step - loss: 1.1388 - accu
Epoch 28/150
1557/1563 [============================>.] - ETA: 0s - loss: 1.1298 - accuracy
Epoch 28: val_accuracy did not improve from 0.51700
1563/1563 [==============================] - 7s 4ms/step - loss: 1.1298 - accu
Epoch 29/150
1559/1563 [============================>.] - ETA: 0s - loss: 1.1153 - accuracy
Epoch 29: val_accuracy did not improve from 0.51700
1563/1563 [==============================] - 6s 4ms/step - loss: 1.1155 - accu
Epoch 30/150
1558/1563 [============================>.] - ETA: 0s - loss: 1.1094 - accuracy
Epoch 30: val_accuracy did not improve from 0.51700
1563/1563 [==============================] - 7s 4ms/step - loss: 1.1094 - accu
Epoch 31/150
1554/1563 [============================>.] - ETA: 0s - loss: 1.1023 - accuracy
Epoch 31: val_accuracy improved from 0.51700 to 0.52380, saving model to best_
1563/1563 [==============================] - 6s 4ms/step - loss: 1.1022 - accu
Epoch 32/150
1563/1563 [==============================] - ETA: 0s - loss: 1.0948 - accuracy
Epoch 32: val_accuracy did not improve from 0.52380
1563/1563 [==============================] - 7s 5ms/step - loss: 1.0948 - accu
Epoch 33/150
1562/1563 [============================>.] - ETA: 0s - loss: 1.0808 - accuracy
Epoch 33: val_accuracy did not improve from 0.52380
1563/1563 [==============================] - 7s 5ms/step - loss: 1.0807 - accu
Epoch 34/150
1556/1563 [============================>.] - ETA: 0s - loss: 1.0723 - accuracy
Epoch 34: val_accuracy did not improve from 0.52380
1563/1563 [==============================] - 6s 4ms/step - loss: 1.0720 - accu
Epoch 35/150
1563/1563 [==============================] - ETA: 0s - loss: 1.0628 - accuracy
Epoch 35: val_accuracy did not improve from 0.52380
1563/1563 [==============================] - 6s 4ms/step - loss: 1.0628 - accu
Epoch 36/150
1550/1563 [============================>.] - ETA: 0s - loss: 1.0595 - accuracy
Epoch 36: val_accuracy did not improve from 0.52380
1563/1563 [==============================] - 7s 4ms/step - loss: 1.0597 - accu
Epoch 37/150
1562/1563 [============================>.] - ETA: 0s - loss: 0.9809 - accuracy
Epoch 37: val_accuracy did not improve from 0.52380
1563/1563 [==============================] - 6s 4ms/step - loss: 0.9808 - accu
```

```
Epoch 38/150
1553/1563 [============================>.] - ETA: 0s - loss: 0.9726 - accuracy
Epoch 38: val_accuracy did not improve from 0.52380
1563/1563 [==============================] - 7s 4ms/step - loss: 0.9727 - accu
Epoch 39/150
1562/1563 [============================>.] - ETA: 0s - loss: 0.9674 - accuracy
Epoch 39: val_accuracy did not improve from 0.52380
1563/1563 [==============================] - 6s 4ms/step - loss: 0.9674 - accu
Epoch 40/150
1558/1563 [============================>.] - ETA: 0s - loss: 0.9564 - accuracy
Epoch 40: val_accuracy improved from 0.52380 to 0.52470, saving model to best_
1563/1563 [==============================] - 7s 4ms/step - loss: 0.9571 - accu
Epoch 41/150
1562/1563 [============================>.] - ETA: 0s - loss: 0.9540 - accuracy
Epoch 41: val_accuracy did not improve from 0.52470
1563/1563 [==============================] - 6s 4ms/step - loss: 0.9542 - accu
Epoch 42/150
1550/1563 [============================>.] - ETA: 0s - loss: 0.9479 - accuracy
Epoch 42: val_accuracy did not improve from 0.52470
1563/1563 [==============================] - 7s 4ms/step - loss: 0.9466 - accu
Epoch 43/150
1549/1563 [============================>.] - ETA: 0s - loss: 0.9408 - accuracy
Epoch 43: val_accuracy improved from 0.52470 to 0.52580, saving model to best_
1563/1563 [==============================] - 6s 4ms/step - loss: 0.9398 - accu
Epoch 44/150
1548/1563 [============================>.] - ETA: 0s - loss: 0.9337 - accuracy
Epoch 44: val_accuracy did not improve from 0.52580
1563/1563 [==============================] - 6s 4ms/step - loss: 0.9337 - accu
Epoch 45/150
1557/1563 [============================>.] - ETA: 0s - loss: 0.9289 - accuracy
Epoch 45: val_accuracy did not improve from 0.52580
1563/1563 [==============================] - 6s 4ms/step - loss: 0.9294 - accu
Epoch 46/150
1552/1563 [============================>.] - ETA: 0s - loss: 0.9230 - accuracy
Epoch 46: val_accuracy did not improve from 0.52580
1563/1563 [==============================] - 7s 4ms/step - loss: 0.9228 - accu
Epoch 47/150
1557/1563 [============================>.] - ETA: 0s - loss: 0.9169 - accuracy
Epoch 47: val_accuracy did not improve from 0.52580
1563/1563 [==============================] - 6s 4ms/step - loss: 0.9174 - accu
Epoch 48/150
1561/1563 [============================>.] - ETA: 0s - loss: 0.9147 - accuracy
Epoch 48: val_accuracy did not improve from 0.52580
1563/1563 [==============================] - 7s 4ms/step - loss: 0.9143 - accu
Epoch 49/150
1553/1563 [============================>.] - ETA: 0s - loss: 0.8697 - accuracy
Epoch 49: val_accuracy improved from 0.52580 to 0.53290, saving model to best_
1563/1563 [==============================] - 7s 4ms/step - loss: 0.8705 - accu
Epoch 50/150
1558/1563 [============================>.] - ETA: 0s - loss: 0.8661 - accuracy
Epoch 50: val_accuracy did not improve from 0.53290
1563/1563 [==============================] - 7s 4ms/step - loss: 0.8662 - accu
```

```
Epoch 51/150
1549/1563 [============================>.] - ETA: 0s - loss: 0.8617 - accurac
Epoch 51: val_accuracy did not improve from 0.53290
1563/1563 [==============================] - 6s 4ms/step - loss: 0.8616 - acc
Epoch 52/150
1556/1563 [============================>.] - ETA: 0s - loss: 0.8586 - accurac
Epoch 52: val_accuracy did not improve from 0.53290
1563/1563 [==============================] - 7s 4ms/step - loss: 0.8584 - acc
Epoch 53/150
1562/1563 [============================>.] - ETA: 0s - loss: 0.8545 - accurac
Epoch 53: val_accuracy did not improve from 0.53290
1563/1563 [==============================] - 6s 4ms/step - loss: 0.8545 - acc
Epoch 54/150
1562/1563 [============================>.] - ETA: 0s - loss: 0.8508 - accurac
Epoch 54: val_accuracy did not improve from 0.53290
1563/1563 [==============================] - 7s 4ms/step - loss: 0.8508 - acc
Epoch 55/150
1560/1563 [============================>.] - ETA: 0s - loss: 0.8407 - accurac
Epoch 55: val_accuracy did not improve from 0.53290
1563/1563 [==============================] - 7s 5ms/step - loss: 0.8404 - acc
Epoch 56/150
1551/1563 [============================>.] - ETA: 0s - loss: 0.8363 - accurac
Epoch 56: val_accuracy did not improve from 0.53290
1563/1563 [==============================] - 8s 5ms/step - loss: 0.8365 - acc
Epoch 57/150
1562/1563 [============================>.] - ETA: 0s - loss: 0.8327 - accurac
Epoch 57: val_accuracy did not improve from 0.53290
1563/1563 [==============================] - 8s 5ms/step - loss: 0.8327 - acc
Epoch 58/150
1557/1563 [============================>.] - ETA: 0s - loss: 0.8301 - accurac
Epoch 58: val_accuracy did not improve from 0.53290
1563/1563 [==============================] - 7s 4ms/step - loss: 0.8301 - acc
Epoch 59/150
1553/1563 [============================>.] - ETA: 0s - loss: 0.8277 - accurac
Epoch 59: val_accuracy did not improve from 0.53290
1563/1563 [==============================] - 8s 5ms/step - loss: 0.8273 - acc
313/313 [==============================] - 1s 3ms/step - loss: 1.5897 - accur
Training Time: 393.0499541759491 seconds
Test Loss: 1.5896819829940796, Test Accuracy: 0.5205000042915344
```







Discussion

The 512 single-layer MLP in the paper has:
a training accuracy of 99.80 and a test accuracy of 50.39 with weight decay a training accuracy of 100.0 and a test accuracy of 50.51 without weight decay

During experimentation with the hyperparameters, we obtained the following results:
For a 1024 single-layer MLP we found a test accuracy of 49.91% without weight decay (lr = 0.001; ADAM; plateau adaptive LR)
For a 2056 single-layer MLP with (lr = 0.005) we found a test accuracy of 39.34%
For a 2056 single-layer MLP with (lr = 0.0005) we found a test accuracy of 48.4% after 150 epochs with severe overfitting. Based on this, we implemented a early stopping mechanism and did another run.

Running this same model again with same parameters but early stopping, we got the curve above. The best model had an accuracy of 53.29% and is stored in the 'best_model.h5' file and can be found in the brightspace submission. It can be loaded using the code at the start of the exercise for inference.

Conclusion:

- Wider layers can learn more by the sheer force of parameters, but are computationally more intensive and in practice not worth it.
- Wide layers are more likely to overfit and need a smaller learning rate to converge properly
- Early stopping can help with a higher patience (10 epochs in our case)

Exercise 3 - Convolutional Neural Networks (CNN)

Study the performance properties of the convolutional network provided in the Tensorflow tutorial. How is the learning affected if instead of ReLU units, tanh() activations are used? What is the reason for this? Compare also at least two different optimizer algorithms.

Create the convolutional base

The 6 lines of code below define the convolutional base using a common pattern: a stack of Conv2D and MaxPooling2D layers.

As input, a CNN takes tensors of shape (image_height, image_width, color_channels), ignoring the batch size. If you are new to these dimensions, color_channels refers to (R,G,B). In this example, you will configure your CNN to process inputs of shape (32, 32, 3), which is the format of CIFAR images. You can do this by passing the argument `input_shape` to your first layer.

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

Let's display the architecture of your model so far:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        896

 max_pooling2d (MaxPooling2  (None, 15, 15, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 13, 13, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 6, 6, 64)          0
 g2D)

 conv2d_2 (Conv2D)           (None, 4, 4, 64)          36928

=================================================================
Total params: 56320 (220.00 KB)
```

```
Trainable params: 56320 (220.00 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Above, you can see that the output of every Conv2D and MaxPooling2D layer is a 3D tensor of shape (height, width, channels). The width and height dimensions tend to shrink as you go deeper in the network. The number of output channels for each Conv2D layer is controlled by the first argument (e.g., 32 or 64). Typically, as the width and height shrink, you can afford (computationally) to add more output channels in each Conv2D layer.

Add Dense layers on top

To complete the model, you will feed the last output tensor from the convolutional base (of shape (4, 4, 64)) into one or more Dense layers to perform classification. Dense layers take vectors as input (which are 1D), while the current output is a 3D tensor. First, you will flatten (or unroll) the 3D output to 1D, then add one or more Dense layers on top. CIFAR has 10 output classes, so you use a final Dense layer with 10 outputs.

```
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))
```

Here's the complete architecture of your model:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        896

 max_pooling2d (MaxPooling2  (None, 15, 15, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 13, 13, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 6, 6, 64)          0
 g2D)

 conv2d_2 (Conv2D)           (None, 4, 4, 64)          36928

 flatten (Flatten)           (None, 1024)              0

 dense (Dense)               (None, 64)                65600
```

```
 dense_1 (Dense)             (None, 10)                650

=================================================================
Total params: 122570 (478.79 KB)
Trainable params: 122570 (478.79 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

The network summary shows that (4, 4, 64) outputs were flattened into vectors of shape (1024) before going through two Dense layers.

Compile and train the model

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))
```

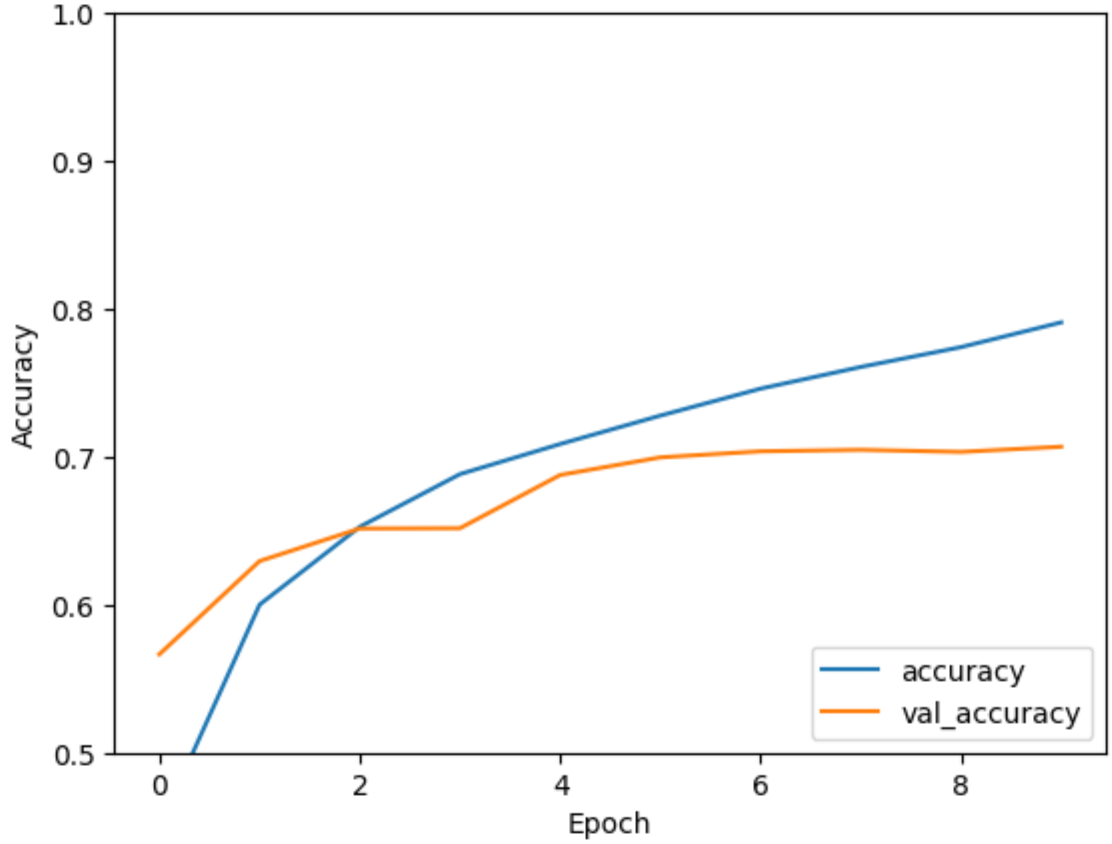
```
Epoch 1/10
1563/1563 [==============================] - 20s 6ms/step - loss: 1.5162 - ac
Epoch 2/10
1563/1563 [==============================] - 8s 5ms/step - loss: 1.1337 - acc
Epoch 3/10
1563/1563 [==============================] - 8s 5ms/step - loss: 0.9906 - acc
Epoch 4/10
1563/1563 [==============================] - 8s 5ms/step - loss: 0.8916 - acc
Epoch 5/10
1563/1563 [==============================] - 7s 5ms/step - loss: 0.8289 - acc
Epoch 6/10
1563/1563 [==============================] - 7s 5ms/step - loss: 0.7747 - acc
Epoch 7/10
1563/1563 [==============================] - 8s 5ms/step - loss: 0.7206 - acc
Epoch 8/10
1563/1563 [==============================] - 8s 5ms/step - loss: 0.6793 - acc
Epoch 9/10
1563/1563 [==============================] - 8s 5ms/step - loss: 0.6390 - acc
Epoch 10/10
1563/1563 [==============================] - 8s 5ms/step - loss: 0.5970 - acc
```

Evaluate the model

```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
```

```
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')

test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
```

```
313/313 - 1s - loss: 0.8816 - accuracy: 0.7068 - 695ms/epoch - 2ms/step
```




```
print(f'Performance of activation relu and oprimizer Adam is: {(test_acc*100):.2f}
```

```
Performance of activation relu and oprimizer Adam is: 70.68 %
```

Your simple CNN has achieved a test accuracy of over 70%. Not bad for a few lines of code! For another CNN style, check out the [TensorFlow 2 quickstart for experts](#) example that uses the Keras subclassing API and `tf.GradientTape`.

Testing with activation = 'tanh'

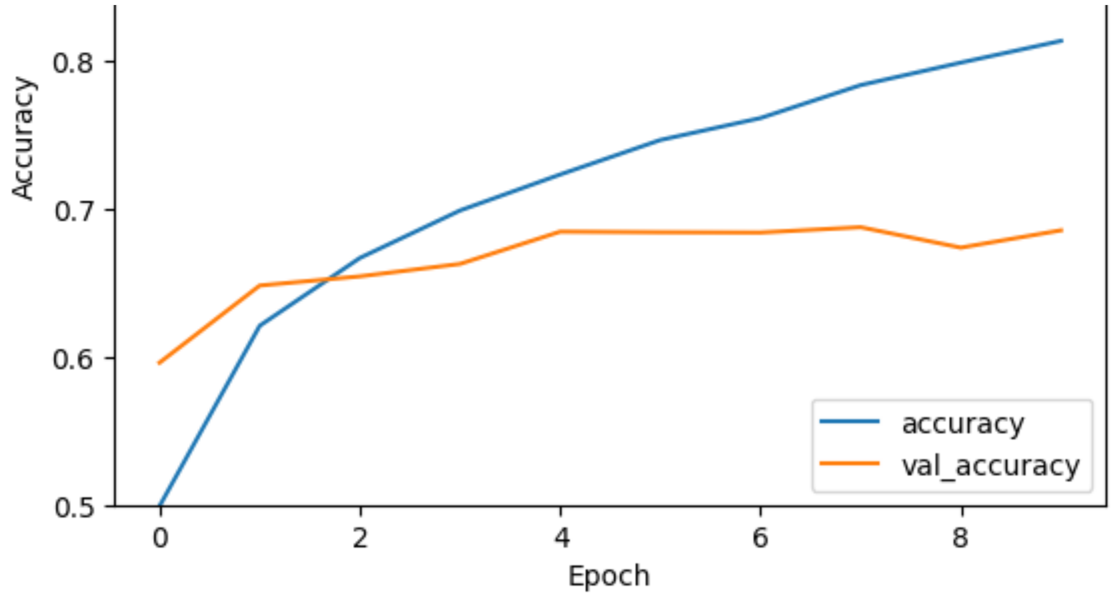
```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='tanh', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='tanh'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='tanh'))
```

```
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))


model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))
```

```
Epoch 1/10
1563/1563 [==============================] - 10s 5ms/step - loss: 1.3969 - ac
Epoch 2/10
1563/1563 [==============================] - 8s 5ms/step - loss: 1.0749 - acc
Epoch 3/10
1563/1563 [==============================] - 8s 5ms/step - loss: 0.9475 - acc
Epoch 4/10
1563/1563 [==============================] - 12s 7ms/step - loss: 0.8593 - ac
Epoch 5/10
1563/1563 [==============================] - 8s 5ms/step - loss: 0.7900 - acc
Epoch 6/10
1563/1563 [==============================] - 8s 5ms/step - loss: 0.7274 - acc
Epoch 7/10
1563/1563 [==============================] - 8s 5ms/step - loss: 0.6717 - acc
Epoch 8/10
1563/1563 [==============================] - 8s 5ms/step - loss: 0.6181 - acc
Epoch 9/10
1563/1563 [==============================] - 8s 5ms/step - loss: 0.5668 - acc
Epoch 10/10
1563/1563 [==============================] - 8s 5ms/step - loss: 0.5198 - acc
```

```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')

test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
```

```
313/313 - 1s - loss: 1.0499 - accuracy: 0.6854 - 696ms/epoch - 2ms/step
```




```
print(f'Performance of activation tanh and oprimizer Adam is: {(test_acc*100):.2f}
```

```
Performance of activation tanh and oprimizer Adam is: 68.54 %
```

How does 'relu' model compare to 'tanh' in performance? What is the reason for this?

The relu model performed better than the tanh one (70.68% compared to 68.54%) as expected when relu will not encounter the Vanishing Gradient problem. This problem is expected to hinder the effective learning in the tanh case. Relu sets all negative values to zero which creates sparcity (similar to the dropout method) reducing computational complexity so further improving the accuracy.

Comparing optimization functions

First, summarizing the already implemented optimizer - the 'adam' optimizer: it combines both the momentum and the root mean squared propagation methods for gradient descent. It maintains the mean and uncentered variance for each parameter and adapts the learning rates for each parameter based on the historical and squared gradients and it includes bias correction.

The second method: 'stochastic gradient descent'. In each iteration, the model updates its parameters using the gradient of the low wrt to a mini-batch of training data and uses a fixed learning rate.

Lastly, the 'root mean square propagation': it maintains a moving average of sqauared gradients for each parameter and the learning rates are scaled inversly proportional to the square root of the average (as opposed to the SGD's fixed learning rate).

Generate the model

```
# generate the model again
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))
```

Stochastic Gradient Descent (SGD)

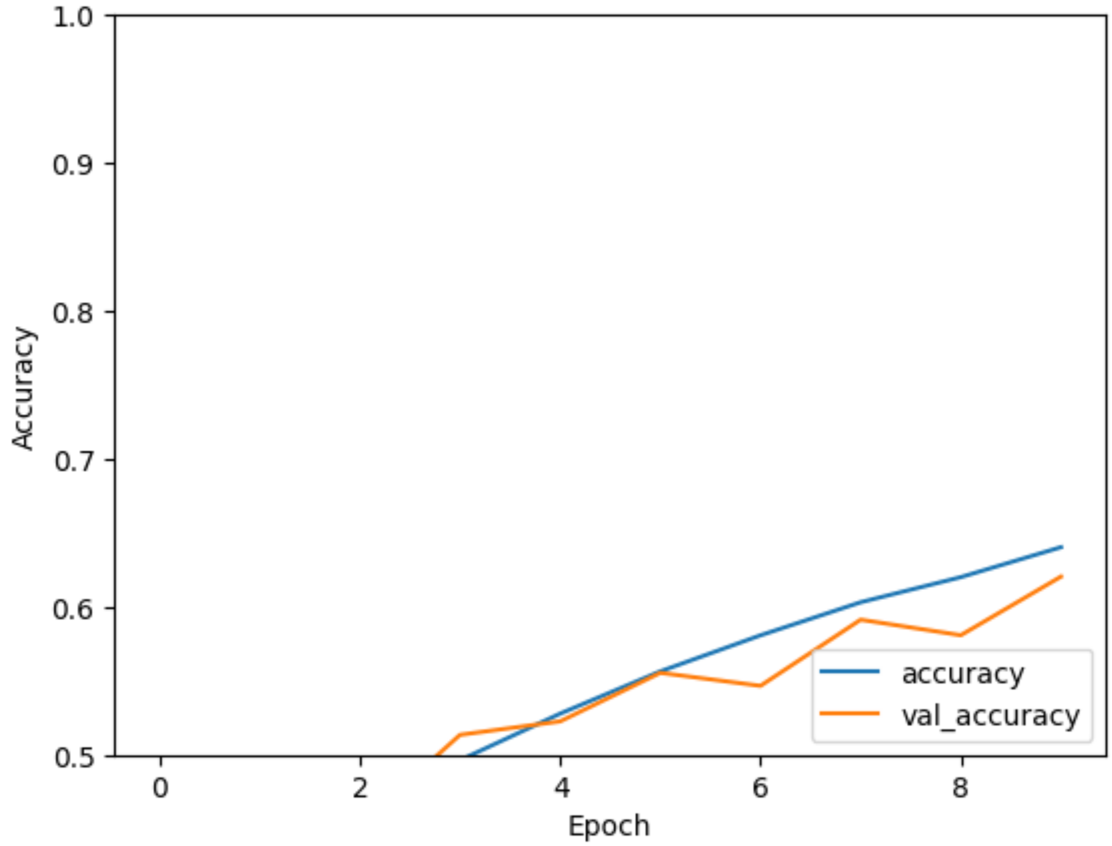
```
# using Stochastic Gradient Descent (SGD) with Momentum as an optiimiser
model.compile(optimizer='SGD',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))
```

```
Epoch 1/10
1563/1563 [==============================] - 9s 5ms/step - loss: 2.0503 - accu
Epoch 2/10
1563/1563 [==============================] - 8s 5ms/step - loss: 1.6974 - accu
Epoch 3/10
1563/1563 [==============================] - 8s 5ms/step - loss: 1.5143 - accu
Epoch 4/10
1563/1563 [==============================] - 8s 5ms/step - loss: 1.4028 - accu
Epoch 5/10
1563/1563 [==============================] - 8s 5ms/step - loss: 1.3206 - accu
Epoch 6/10
1563/1563 [==============================] - 8s 5ms/step - loss: 1.2471 - accu
Epoch 7/10
1563/1563 [==============================] - 7s 5ms/step - loss: 1.1816 - accu
Epoch 8/10
1563/1563 [==============================] - 7s 4ms/step - loss: 1.1252 - accu
Epoch 9/10
1563/1563 [==============================] - 8s 5ms/step - loss: 1.0731 - accu
Epoch 10/10
1563/1563 [==============================] - 8s 5ms/step - loss: 1.0265 - accu
```

```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
```

```
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')

test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
```

```
313/313 - 1s - loss: 1.0881 - accuracy: 0.6204 - 673ms/epoch - 2ms/step
```




```
print(f'Performance of activation relu and oprimizer Stochastic Gradient Descent is
```

```
Performance of activation relu and oprimizer Stochastic Gradient Descent is: (
```

Root mean squared propagation

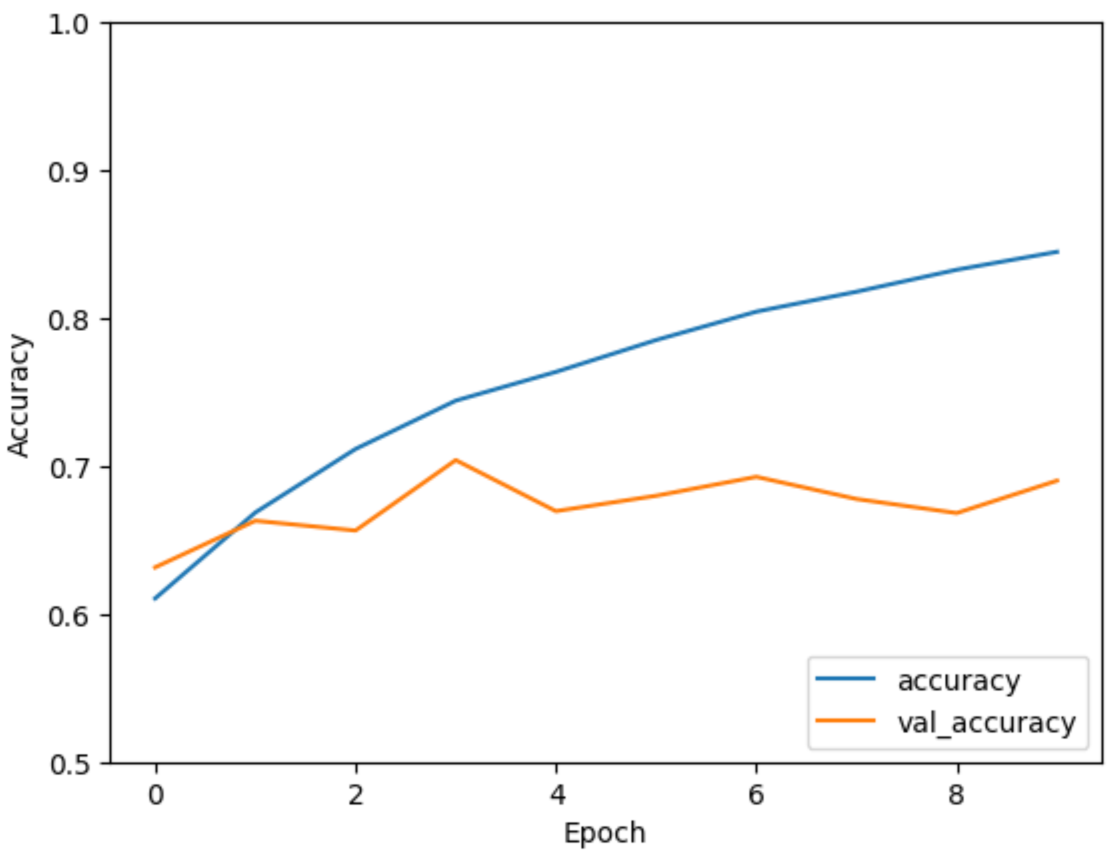
```
# using RMSprop as optimiser algorithm
model.compile(optimizer='RMSprop',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))
```

```
Epoch 1/10
1563/1563 [==============================] - 10s 5ms/step - loss: 1.1157 - ac
Epoch 2/10
1563/1563 [==============================] - 8s 5ms/step - loss: 0.9470 - acc
```

```
1563/1563 [==============================] - 8s 5ms/step - loss: 0.9470 - accu
Epoch 3/10
1563/1563 [==============================] - 7s 5ms/step - loss: 0.8311 - accu
Epoch 4/10
1563/1563 [==============================] - 8s 5ms/step - loss: 0.7481 - accu
Epoch 5/10
1563/1563 [==============================] - 8s 5ms/step - loss: 0.6804 - accu
Epoch 6/10
1563/1563 [==============================] - 8s 5ms/step - loss: 0.6187 - accu
Epoch 7/10
1563/1563 [==============================] - 8s 5ms/step - loss: 0.5696 - accu
Epoch 8/10
1563/1563 [==============================] - 7s 5ms/step - loss: 0.5291 - accu
Epoch 9/10
1563/1563 [==============================] - 8s 5ms/step - loss: 0.4868 - accu
Epoch 10/10
1563/1563 [==============================] - 8s 5ms/step - loss: 0.4538 - accu
```

```python
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')

test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
```

```
313/313 - 1s - loss: 1.0202 - accuracy: 0.6901 - 672ms/epoch - 2ms/step
```



```
print(f'Performance of activation relu and oprimizer Root Mean Squared Propagation
```

```
    Performance of activation relu and oprimizer Root Mean Squared Propagation is
```

Discussion

As expected, the adaptive learning rates from the adam (70.68%) and RMSprop optimizers (69.01%) led to a much higher performance compared to the SGD optimizer (62.04%). Both RMSprop and adam use momentum and are less sensitive to initial weights and learning rate and can also handle noisier data. Thuse, they are more well suited for an image recognition task as given in this assignment.

However, if we were to have for example some storage restrictions or our models would seem unstable or overfit, SGD could be a more reliable choice.

Exercise 4 - CNN vs MLP

Try to outperform the convolutional network of part 3 with a MLP that uses approximately the same number of parameters. The CNN had a total of 122570 parameters.

We will create two models with approximately the same number of parameters with different structures and see if we can outperform the CNN. Report your results and explain them.

We can calculate the number of free parameters by:

(#neurons_first_hidden_layer * #neurons_second_hidden_layer) + (#neurons_second_hidden_layer * #neurons_third_hidden_layer) + ... + (#neurons_x_hidden_layer * #neurons_last_hidden_layer) + (#neurons_in_first_hidden_layer + ... + #neurons_in_last_hidden_layer) = #free_parameters

Therefore if we have a model with three hidden layers of 896, 128, and 64 we get

(896 * 128) + (128 * 64) + (64 * 10) + (896 + 128 + 64 + 10) = 124618

another option is:

(256 * 256)+(256 * 128)+(128 * 128)+(128 * 64)+(64 * 10)+(256+256+128+128+64+10) = 124362

```
#To create a MLP with approximately the same number of parameters as the CNN,
#we create hidden layers with the number of parameters described above in model sum

model_3 = models.Sequential([
    layers.Flatten(),                       # input layer [flatten image to 1D vector
    layers.Dense(896, activation='relu') # hidden layer with 896 neurons and relu a
```

```
    layers.Dense(896, activation='relu'),# hidden layer with 896 neurons and relu a
    layers.Dense(128, activation='relu'),# hidden layer with 128 neurons and relu a
    layers.Dense(64, activation='relu'),# hidden layer with 64 neurons and relu act
    layers.Dense(10, activation='softmax')# output layer of 10 catergories with sof
])

model_4 = models.Sequential([
    layers.Flatten(),                    # input layer [flatten image to 1D vector
    layers.Dense(256, activation='relu'),# hidden layer with 896 neurons and relu a
    layers.Dense(256, activation='relu'),# hidden layer with 128 neurons and relu a
    layers.Dense(128, activation='relu'),# hidden layer with 64 neurons and relu ac
    layers.Dense(128, activation='relu'),# hidden layer with 64 neurons and relu ac
    layers.Dense(64, activation='relu'),# hidden layer with 64 neurons and relu act
    layers.Dense(10, activation='softmax')# output layer of 10 catergories with sof
])


np.random.seed(0)
tf.random.set_seed(0)

model_3.compile(optimizer='adam',
             loss='sparse_categorical_crossentropy',
             metrics=['accuracy'])

model_4.compile(optimizer='adam',
             loss='sparse_categorical_crossentropy',
             metrics=['accuracy'])
```

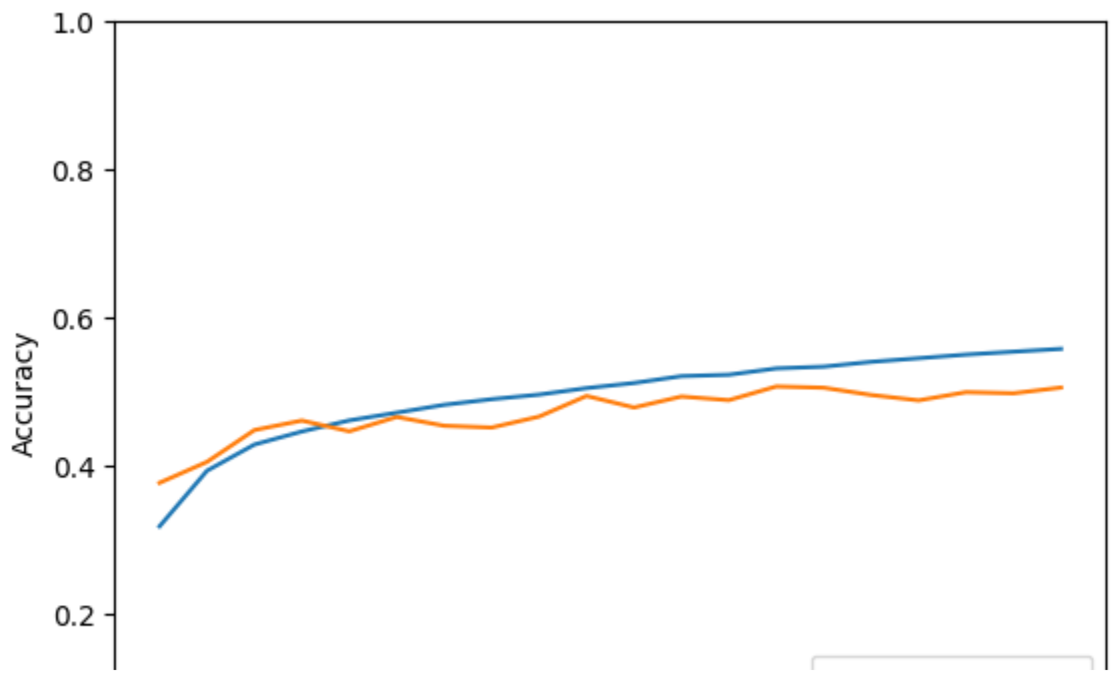
```
history_MLP = model_3.fit(train_images, train_labels, epochs=20,
                    validation_data=(test_images, test_labels))
```

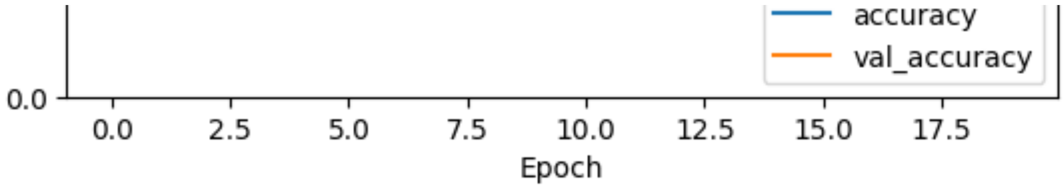
```
Epoch 1/20
1563/1563 [==============================] - 16s 6ms/step - loss: 1.8911 - ac
Epoch 2/20
1563/1563 [==============================] - 7s 4ms/step - loss: 1.6889 - acc
Epoch 3/20
1563/1563 [==============================] - 6s 4ms/step - loss: 1.5987 - acc
Epoch 4/20
1563/1563 [==============================] - 7s 5ms/step - loss: 1.5472 - acc
Epoch 5/20
1563/1563 [==============================] - 6s 4ms/step - loss: 1.5074 - acc
Epoch 6/20
1563/1563 [==============================] - 7s 4ms/step - loss: 1.4805 - acc
Epoch 7/20
1563/1563 [==============================] - 6s 4ms/step - loss: 1.4531 - acc
Epoch 8/20
1563/1563 [==============================] - 7s 4ms/step - loss: 1.4285 - acc
Epoch 9/20
1563/1563 [==============================] - 6s 4ms/step - loss: 1.4064 - acc
Epoch 10/20
```

```
1563/1563 [==============================] - 7s 4ms/step - loss: 1.3830 - accu
Epoch 11/20
1563/1563 [==============================] - 6s 4ms/step - loss: 1.3622 - accu
Epoch 12/20
1563/1563 [==============================] - 7s 5ms/step - loss: 1.3444 - accu
Epoch 13/20
1563/1563 [==============================] - 6s 4ms/step - loss: 1.3321 - accu
Epoch 14/20
1563/1563 [==============================] - 7s 5ms/step - loss: 1.3120 - accu
Epoch 15/20
1563/1563 [==============================] - 7s 4ms/step - loss: 1.2976 - accu
Epoch 16/20
1563/1563 [==============================] - 6s 4ms/step - loss: 1.2842 - accu
Epoch 17/20
1563/1563 [==============================] - 7s 5ms/step - loss: 1.2692 - accu
Epoch 18/20
1563/1563 [==============================] - 6s 4ms/step - loss: 1.2545 - accu
Epoch 19/20
1563/1563 [==============================] - 7s 4ms/step - loss: 1.2409 - accu
Epoch 20/20
1563/1563 [==============================] - 6s 4ms/step - loss: 1.2339 - accu
```

```
plt.plot(history_MLP.history['accuracy'], label='accuracy')
plt.plot(history_MLP.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')

MLP_test_loss, MLP_test_acc = model_3.evaluate(test_images,  test_labels, verbose=2
print(f'Performance of MLP with the first cofiguration is: {(MLP_test_acc*100):.2f}
```

```
313/313 - 1s - loss: 1.4204 - accuracy: 0.5051 - 882ms/epoch - 3ms/step
Performance of MLP with the first cofiguration is: 50.51 %
```





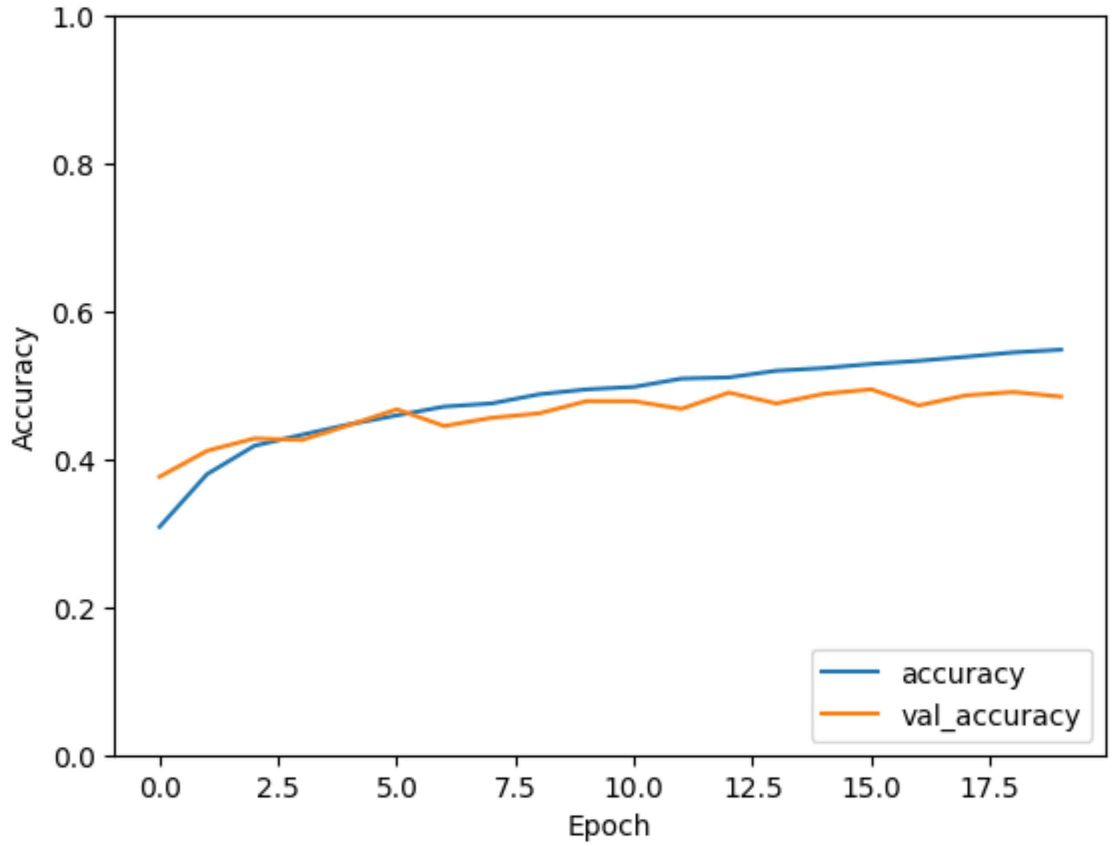

```
history_MLP = model_4.fit(train_images, train_labels, epochs=20,
                          validation_data=(test_images, test_labels))
```

```
Epoch 1/20
1563/1563 [==============================] - 10s 5ms/step - loss: 1.8911 - ac
Epoch 2/20
1563/1563 [==============================] - 6s 4ms/step - loss: 1.7086 - acc
Epoch 3/20
1563/1563 [==============================] - 7s 4ms/step - loss: 1.6230 - acc
Epoch 4/20
1563/1563 [==============================] - 6s 4ms/step - loss: 1.5776 - acc
Epoch 5/20
1563/1563 [==============================] - 7s 5ms/step - loss: 1.5402 - acc
Epoch 6/20
1563/1563 [==============================] - 6s 4ms/step - loss: 1.5061 - acc
Epoch 7/20
1563/1563 [==============================] - 7s 5ms/step - loss: 1.4769 - acc
Epoch 8/20
1563/1563 [==============================] - 7s 5ms/step - loss: 1.4589 - acc
Epoch 9/20
1563/1563 [==============================] - 7s 5ms/step - loss: 1.4296 - acc
Epoch 10/20
1563/1563 [==============================] - 7s 5ms/step - loss: 1.4081 - acc
Epoch 11/20
1563/1563 [==============================] - 7s 4ms/step - loss: 1.3947 - acc
Epoch 12/20
1563/1563 [==============================] - 8s 5ms/step - loss: 1.3725 - acc
Epoch 13/20
1563/1563 [==============================] - 7s 5ms/step - loss: 1.3589 - acc
Epoch 14/20
1563/1563 [==============================] - 7s 4ms/step - loss: 1.3410 - acc
Epoch 15/20
1563/1563 [==============================] - 7s 5ms/step - loss: 1.3303 - acc
Epoch 16/20
1563/1563 [==============================] - 6s 4ms/step - loss: 1.3086 - acc
Epoch 17/20
1563/1563 [==============================] - 7s 5ms/step - loss: 1.2978 - acc
Epoch 18/20
1563/1563 [==============================] - 7s 4ms/step - loss: 1.2828 - acc
Epoch 19/20
1563/1563 [==============================] - 7s 5ms/step - loss: 1.2684 - acc
Epoch 20/20
1563/1563 [==============================] - 7s 4ms/step - loss: 1.2574 - acc
```

```
plt.plot(history_MLP.history['accuracy'], label='accuracy')
plt.plot(history_MLP.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
```

```
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')

MLP_test_loss, MLP_test_acc = model_4.evaluate(test_images,  test_labels, verbose=2
print(f'Performance of MLP with the second cofiguration is: {(MLP_test_acc*100):.2f
```

```
313/313 - 1s - loss: 1.4836 - accuracy: 0.4847 - 897ms/epoch - 3ms/step
Performance of MLP with the second cofiguration is: 48.47 %
```



Discussion:

As shown below, the MLP cannot outperform the CNN. After all, the CNN was made for these types of problems. It is possible to get different cofigurations that lead to approximately the same number of parameters as in the CNN. However, MLP cannot get much higher than a 0.5 validation accuracy with both the tested configurations here. With the CNN this was quickly at 0.7 as demonstrated earlier.