Task3: Vision Transformers for End-to-End Particle Identification with the CMS Experiment

- Datasets:Same as in Task 1
- Description:
  - Train a Transformer model of your choice on the dataset below to achieve the performance closest to your CNN model's performance in Task 1.
  - Discuss the resulting performance of the 2 chosen architectures.

Setup

```
import h5py
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import tensorflow_addons as tfa
```

Data Preparation

```
f1 = h5py.File('../input/electron-photon/download', 'r')
f2 = h5py.File('../input/electron-photon/download_1', 'r')

Electron_X = np.array(f1['X'])
Electron_y = np.array(f1['y'])
Parton_X = np.array(f2['X'])
Parton_y = np.array(f2['y'])
print(Electron_X.shape, Electron_y.shape, Parton_X.shape, Parton_y.shape)

All_X = np.concatenate((Electron_X, Parton_X), axis=0)
All_y = np.concatenate((Electron_y, Parton_y), axis=0)
print(All_X.shape, All_y.shape)
# Then, randomly shuffle the data and split data for train, val, test
rand_seed = 12
index = np.random.permutation(len(All_y))
All_X, All_y = All_X[index][:,:,:,0], All_y[index]
print(All_X.shape, All_y.shape)

# clear cache to save memory
del Electron_X, Electron_y, Parton_X, Parton_y

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(All_X, All_y, test_size=0.2, random_state=12)
X_train = X_train.reshape((-1,32,32,1))
X_test = X_test.reshape((-1,32,32,1))
print(X_train.shape, X_test.shape)
print(y_train.shape, y_test.shape)

del All_X, All_y
```

```
(249000, 32, 32, 2) (249000,) (249000, 32, 32, 2) (249000,)
(498000, 32, 32, 2) (498000,)
(498000, 32, 32) (498000,)
(398400, 32, 32, 1) (99600, 32, 32, 1)
(398400,) (99600,)
```

Hyperparameters Configuration

```
num_classes = 1
input_shape = (32, 32, 1)
learning_rate = 0.0015
batch_size = 64
num_epochs = 50
image_size = 32 # size for resize image
patch_size = 8  # size of the patches to be extract from the input images
num_patches = (image_size // patch_size) ** 2
projection_dim = 64
num_heads = 4
transformer_units = [
    projection_dim * 2,
    projection_dim,
]  # Size of the transformer layers
transformer_layers = 1
mlp_head_units = [1024, 512]  # Size of the dense layers of the final classifier
```

```
data_augmentation = keras.Sequential(
    [
        layers.Normalization(),
        layers.Resizing(image_size, image_size),
    ],
    name="data_augmentation",
)
data_augmentation.layers[0].adapt(X_train)
```

```
2022-03-25 13:23:58.526401: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] success
2022-03-25 13:23:58.622269: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] success
2022-03-25 13:23:58.623332: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] success
2022-03-25 13:23:58.626663: I tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlo
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2022-03-25 13:23:58.627068: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] success
2022-03-25 13:23:58.627896: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] success
2022-03-25 13:23:58.628610: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] success
2022-03-25 13:24:00.555412: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] success
2022-03-25 13:24:00.556465: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] success
2022-03-25 13:24:00.557235: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] success
2022-03-25 13:24:00.558850: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1510] Created de
2022-03-25 13:24:00.906725: W tensorflow/core/framework/cpu_allocator_impl.cc:80] Allocation of
2022-03-25 13:24:02.626296: W tensorflow/core/framework/cpu_allocator_impl.cc:80] Allocation of
2022-03-25 13:24:03.827638: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] Non
```

ViT model from keras code example

```
class Patches(layers.Layer):
    def __init__(self, patch_size):
```

```
        super(Patches, self).__init__()
        self.patch_size = patch_size

    def call(self, images):
        batch_size = tf.shape(images)[0]
        patches = tf.image.extract_patches(
            images=images,
            sizes=[1, self.patch_size, self.patch_size, 1],
            strides=[1, self.patch_size, self.patch_size, 1],
            rates=[1, 1, 1, 1],
            padding="VALID",
        )
        patch_dims = patches.shape[-1]
        patches = tf.reshape(patches, [batch_size, -1, patch_dims])
        return patches

class PatchEncoder(layers.Layer):
    def __init__(self, num_patches, projection_dim):
        super(PatchEncoder, self).__init__()
        self.num_patches = num_patches
        self.projection = layers.Dense(units=projection_dim)
        self.position_embedding = layers.Embedding(
            input_dim=num_patches, output_dim=projection_dim
        )

    def call(self, patch):
        positions = tf.range(start=0, limit=self.num_patches, delta=1)
        encoded = self.projection(patch) + self.position_embedding(positions)
        return encoded
```

```
def mlp(x, hidden_units, dropout_rate):
    for units in hidden_units:
        x = layers.Dense(units, activation=tf.nn.gelu)(x)
        x = layers.Dropout(dropout_rate)(x)
    return x

def create_vit_classifier():
    inputs = layers.Input(shape=input_shape)
    # Augment data.
    augmented = data_augmentation(inputs)
    # Create patches.
    patches = Patches(patch_size)(augmented)
    # Encode patches.
    encoded_patches = PatchEncoder(num_patches, projection_dim)(patches)

    # Create multiple layers of the Transformer block.
    for _ in range(transformer_layers):
        # Layer normalization 1.
        x1 = layers.LayerNormalization(epsilon=1e-6)(encoded_patches)
        # Create a multi-head attention layer.
        attention_output = layers.MultiHeadAttention(
            num_heads=num_heads, key_dim=projection_dim, dropout=0.1
        )(x1, x1)
        # Skip connection 1.
        x2 = layers.Add()([attention_output, encoded_patches])
        # Layer normalization 2.
        x3 = layers.LayerNormalization(epsilon=1e-6)(x2)
        # MLP.
        x3 = mlp(x3, hidden_units=transformer_units, dropout_rate=0.1)
        # Skip connection 2.
        encoded_patches = layers.Add()([x3, x2])
```

```
    # Create a [batch_size, projection_dim] tensor.
    representation = layers.LayerNormalization(epsilon=1e-6)(encoded_patches)
    representation = layers.Flatten()(representation)
    representation = layers.Dropout(0.5)(representation)
    # Add MLP.
    features = mlp(representation, hidden_units=mlp_head_units, dropout_rate=0.5)
    # Classify outputs.
    logits = layers.Dense(num_classes)(features)
    # Create the Keras model.
    model = keras.Model(inputs=inputs, outputs=logits)
    return model
```

Model Compiling and Training

```
model = create_vit_classifier()

optimizer = tf.keras.optimizers.Adam(
    learning_rate=learning_rate
)
model.compile(
    optimizer=optimizer,
    loss=keras.losses.BinaryCrossentropy(from_logits=True),
    metrics=[
        keras.metrics.BinaryAccuracy(name="binary_accuracy", dtype=None, threshold=0.5),
        keras.metrics.AUC(from_logits=True),
    ],
)

checkpoint_filepath = "saved_model"
checkpoint_callback = keras.callbacks.ModelCheckpoint(
    checkpoint_filepath,
    monitor="val_binary_accuracy",
    save_best_only=True,
    save_weights_only=True,
)

history = model.fit(
    x=X_train,
    y=y_train,
    batch_size=batch_size,
    epochs=num_epochs,
    validation_split=0.2,
    callbacks=[checkpoint_callback],
    shuffle=True,
)
```

```
Epoch 23/50
4980/4980 [==============================] - 38s 8ms/step - loss: 0.5675 - binary_accuracy:
Epoch 24/50
4980/4980 [==============================] - 35s 7ms/step - loss: 0.5656 - binary_accuracy:
Epoch 25/50
4980/4980 [==============================] - 39s 8ms/step - loss: 0.5663 - binary_accuracy:
Epoch 26/50
4980/4980 [==============================] - 36s 7ms/step - loss: 0.5659 - binary_accuracy:
Epoch 27/50
4980/4980 [==============================] - 40s 8ms/step - loss: 0.5657 - binary_accuracy:
Epoch 28/50
4980/4980 [==============================] - 36s 7ms/step - loss: 0.5667 - binary_accuracy:
Epoch 29/50
4980/4980 [==============================] - 39s 8ms/step - loss: 0.5663 - binary_accuracy:
```

```
4980/4980 [==============================] - 39s 8ms/step - loss: 0.5665 - binary_accuracy:
Epoch 30/50
4980/4980 [==============================] - 36s 7ms/step - loss: 0.5666 - binary_accuracy:
Epoch 31/50
4980/4980 [==============================] - 39s 8ms/step - loss: 0.5644 - binary_accuracy:
Epoch 32/50
4980/4980 [==============================] - 36s 7ms/step - loss: 0.5673 - binary_accuracy:
Epoch 33/50
4980/4980 [==============================] - 40s 8ms/step - loss: 0.5653 - binary_accuracy:
Epoch 34/50
4980/4980 [==============================] - 35s 7ms/step - loss: 0.5691 - binary_accuracy:
Epoch 35/50
4980/4980 [==============================] - 41s 8ms/step - loss: 0.5635 - binary_accuracy:
Epoch 36/50
4980/4980 [==============================] - 36s 7ms/step - loss: 0.5663 - binary_accuracy:
Epoch 37/50
4980/4980 [==============================] - 40s 8ms/step - loss: 0.5649 - binary_accuracy:
Epoch 38/50
4980/4980 [==============================] - 35s 7ms/step - loss: 0.5700 - binary_accuracy:
Epoch 39/50

4980/4980 [==============================] - 40s 8ms/step - loss: 0.5633 - binary_accuracy:
Epoch 40/50
4980/4980 [==============================] - 35s 7ms/step - loss: 0.5726 - binary_accuracy:
Epoch 41/50
4980/4980 [==============================] - 35s 7ms/step - loss: 0.5639 - binary_accuracy:
Epoch 42/50
4980/4980 [==============================] - 38s 8ms/step - loss: 0.5656 - binary_accuracy:
Epoch 43/50
4980/4980 [==============================] - 36s 7ms/step - loss: 0.5666 - binary_accuracy:
Epoch 44/50
4980/4980 [==============================] - 35s 7ms/step - loss: 0.5632 - binary_accuracy:
Epoch 45/50
4980/4980 [==============================] - 35s 7ms/step - loss: 0.5709 - binary_accuracy:
Epoch 46/50
4980/4980 [==============================] - 41s 8ms/step - loss: 0.5667 - binary_accuracy:
Epoch 47/50
4980/4980 [==============================] - 36s 7ms/step - loss: 0.5616 - binary_accuracy:
Epoch 48/50
4980/4980 [==============================] - 37s 7ms/step - loss: 0.5666 - binary_accuracy:
Epoch 49/50
4980/4980 [==============================] - 35s 7ms/step - loss: 0.5703 - binary_accuracy:
Epoch 50/50
4980/4980 [==============================] - 42s 8ms/step - loss: 0.5692 - binary_accuracy:
```

```
!ls
```

```
__notebook__.ipynb  saved_model.data-00000-of-00001
checkpoint          saved_model.index
```

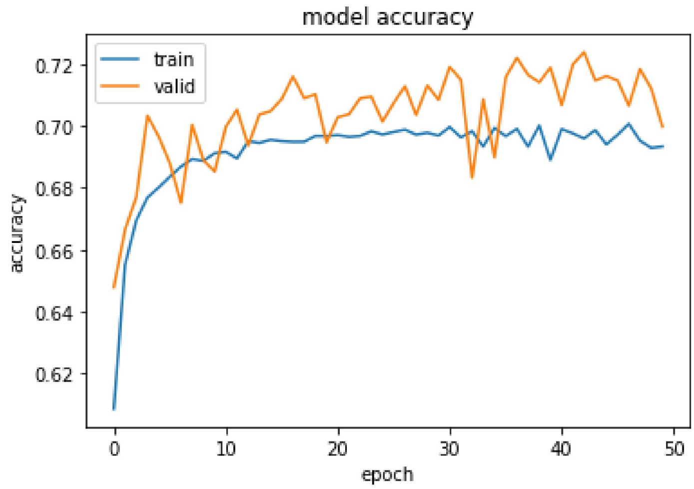
```
model.load_weights(checkpoint_filepath)
_, accuracy, auc = model.evaluate(X_test, y_test)
print(f"Test accuracy: {accuracy}")
print(f"Test AUC: {auc}")
```

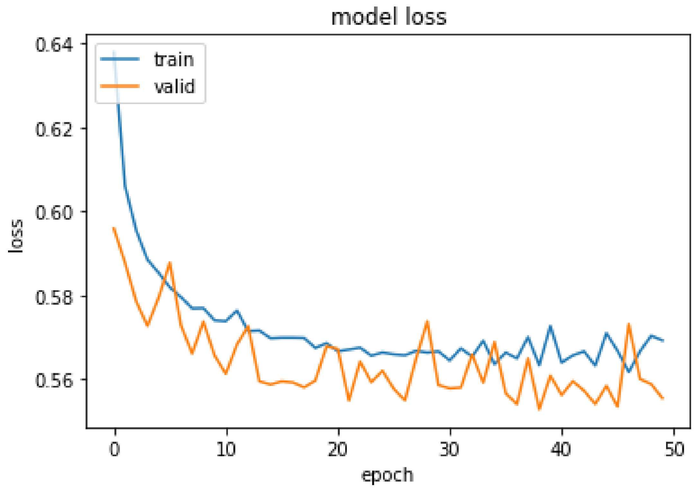
```
3113/3113 [==============================] - 18s 5ms/step - loss: 0.5560 - binary_accuracy: 0.7
Test accuracy: 0.7228614687919617
Test AUC: 0.7942342162132263
```

Plot Training Process

```
import matplotlib.pyplot as plt
print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['binary_accuracy'])
plt.plot(history.history['val_binary_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'valid'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'valid'], loc='upper left')
plt.show()
```

```
dict_keys(['loss', 'binary_accuracy', 'auc', 'val_loss', 'val_binary_accuracy', 'val_auc'])
```






```
model.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 32, 32, 1)]  0
__________________________________________________________________________________________________
data_augmentation (Sequential)  (None, 32, 32, 1)    3           input_1[0][0]
```

```
patches (Patches)              (None, None, 64)     0           data_augmentation[0][0]

patch_encoder (PatchEncoder)   (None, 16, 64)       5184        patches[0][0]

layer_normalization (LayerNorma (None, 16, 64)      128         patch_encoder[0][0]

multi_head_attention (MultiHead (None, 16, 64)      66368       layer_normalization[0][0]
                                                                layer_normalization[0][0]

add (Add)                      (None, 16, 64)       0           multi_head_attention[0][0]
                                                                patch_encoder[0][0]

layer_normalization_1 (LayerNor (None, 16, 64)      128         add[0][0]

dense_1 (Dense)                (None, 16, 128)      8320        layer_normalization_1[0][0]

dropout (Dropout)              (None, 16, 128)      0           dense_1[0][0]

dense_2 (Dense)                (None, 16, 64)       8256        dropout[0][0]

dropout_1 (Dropout)            (None, 16, 64)       0           dense_2[0][0]

add_1 (Add)                    (None, 16, 64)       0           dropout_1[0][0]
                                                                add[0][0]

layer_normalization_2 (LayerNor (None, 16, 64)      128         add_1[0][0]

flatten (Flatten)              (None, 1024)         0           layer_normalization_2[0][0]

dropout_2 (Dropout)            (None, 1024)         0           flatten[0][0]

dense_3 (Dense)                (None, 1024)         1049600     dropout_2[0][0]

dropout_3 (Dropout)            (None, 1024)         0           dense_3[0][0]

dense_4 (Dense)                (None, 512)          524800      dropout_3[0][0]

dropout_4 (Dropout)            (None, 512)          0           dense_4[0][0]

dense_5 (Dense)                (None, 1)            513         dropout_4[0][0]
==================================================================================================
Total params: 1,663,428
Trainable params: 1,663,425
Non-trainable params: 3
```