

## ДОДАТОК А

### Вигляд створених архітектур

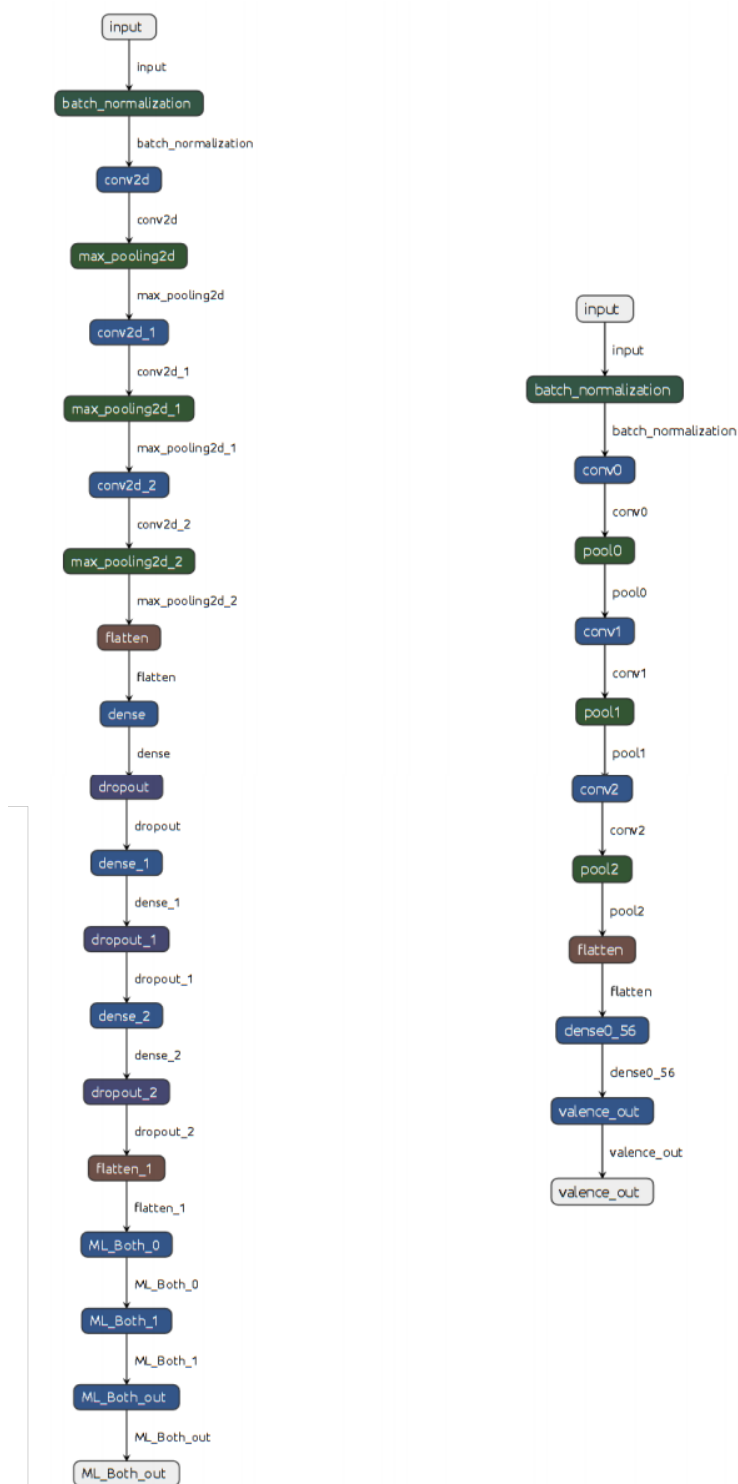


Рисунок А.1 – Представлення архітектури SLN для розглянутих датасетів

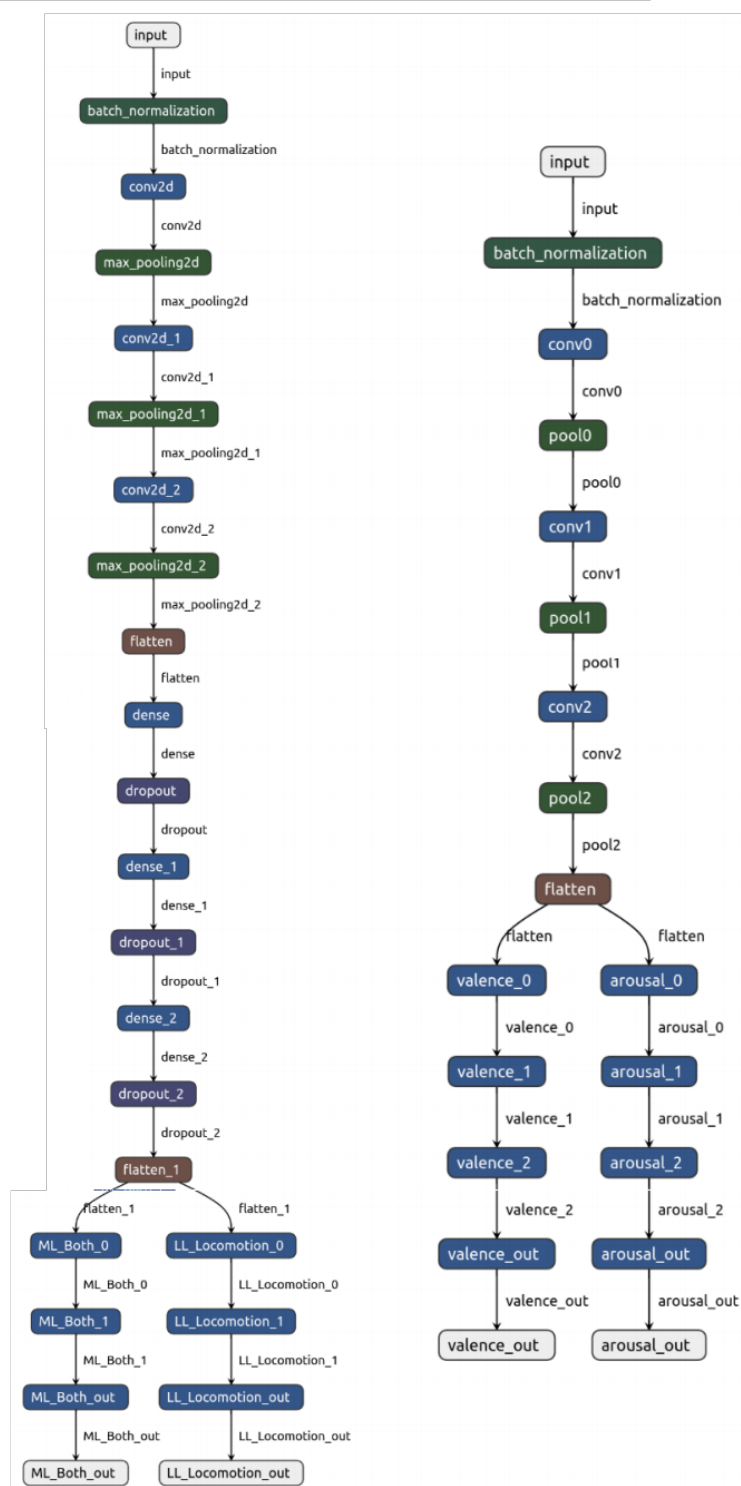


Рисунок А.2 – Представлення архітектури HPS для розглянутих датасетів

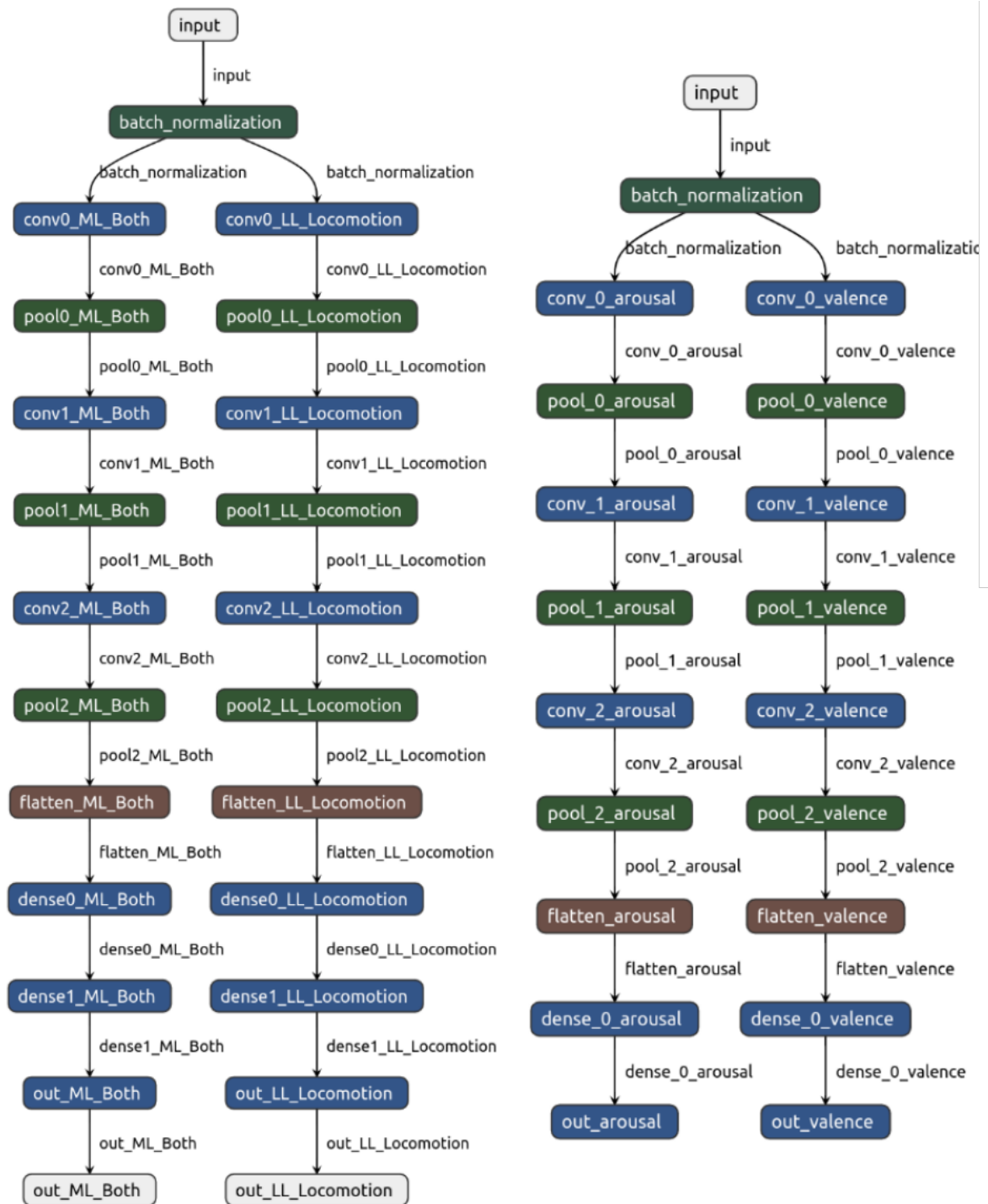


Рисунок А.3 – Представлення архітектури SPS для розглянутих датасетів

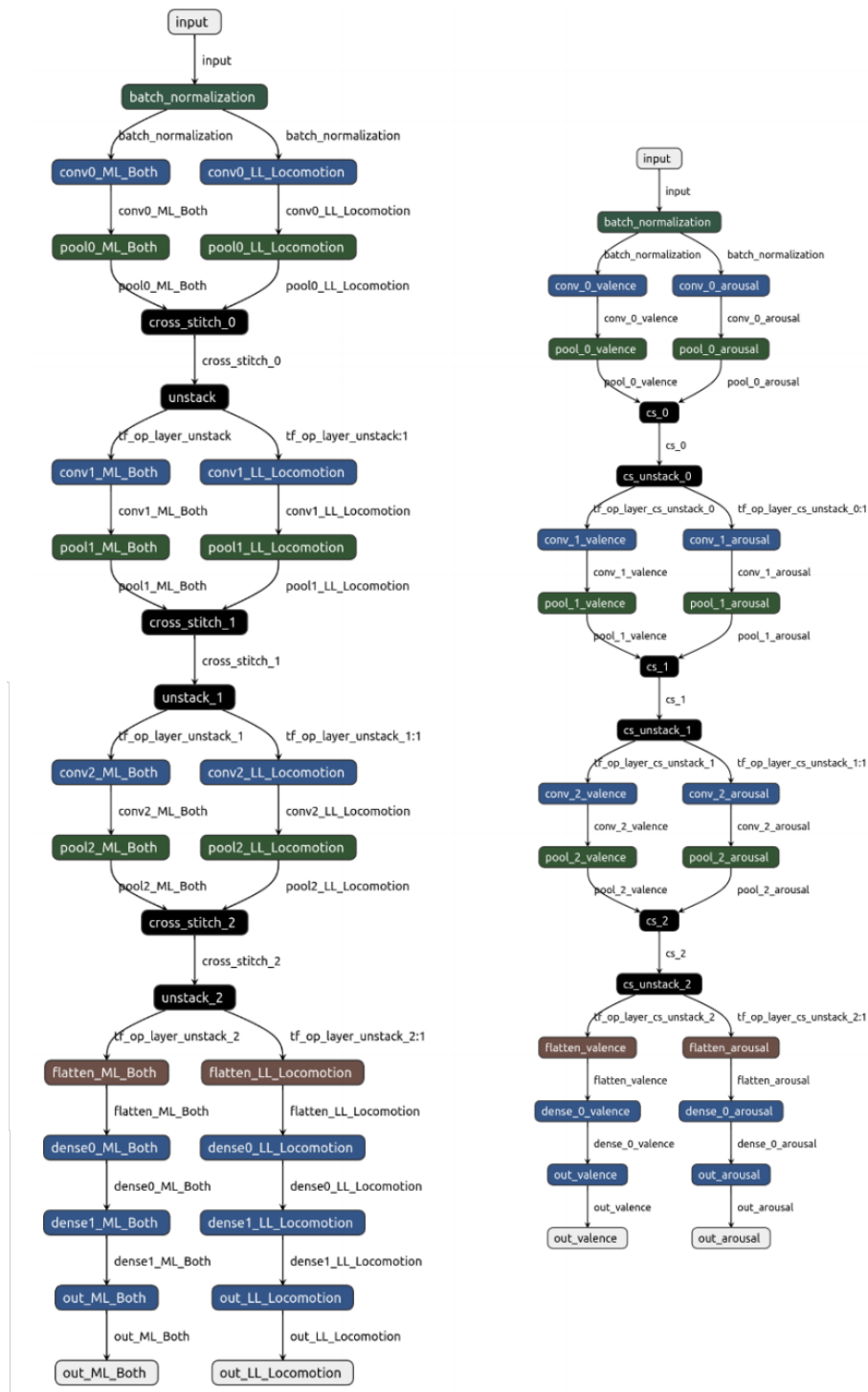


Рисунок А.4 – Представлення архітектури SPS для розглянутих датасетів

## ДОДАТОК Б

### Програмний код

#### Лістинг Б.1 – Програмний код тензорної норми сліду

```

def loss_trace_norm(self, ttn_layers, base_fun,
ttn_weight=None):

    def loss(y_true, y_pred):
        def _ttn(layer_idx, ttn_layer):
            weights =
[self.model.get_layer(name=conv_name).kernel for conv_name in
ttn_layer]

            shapeX = weights[0].get_shape().as_list()
            dimX = len(shapeX)
            stack = tf.stack(weights, axis=dimX)
            t = TensorTraceNorm(stack)
            tr = tf.reduce_sum(t)
            return tr

        cce = base_fun(y_true, y_pred)
        ttn = [_ttn(layer_idx, ttn_layer) for layer_idx,
ttn_layer in enumerate(ttn_layers)]
        sttn = tf.reduce_sum(ttn)

        if ttn_weight is not None:
            sttn = tf.math.scalar_mul(ttn_weight, sttn)

        final_loss = cce + sttn
        return final_loss

    if ttn_layers is None:
        return base_fun

    return loss

```

## Лістинг Б.2 – Програмний код класу Cross-Stitch Network

```

class CrossStitch(tf.keras.layers.Layer):

    def __init__(self, num_tasks, *args, **kwargs):
        self.num_tasks = num_tasks
        super(CrossStitch, self).__init__(**kwargs)

    def build(self, input_shape):
        self.kernel = self.add_weight(
            name="kernel",
            shape=(self.num_tasks, self.num_tasks),
            initializer='identity',
            trainable=True
        )
        super(CrossStitch, self).build(input_shape)

    def call(self, x1):
        if len(x1) != self.num_tasks:
            raise ValueError()

        out_values = []
        for this_task in range(self.num_tasks):
            this_weight = self.kernel[this_task,
this_task]

            out = tf.math.scalar_mul(this_weight,
x1[this_task])

            for other_task in range(self.num_tasks):
                if this_task == other_task:
                    continue
                other_weight = self.kernel[this_task,
other_task]

                out += tf.math.scalar_mul(other_weight,
x1[other_task])

```

```

        out_values.append(out)

    return tf.stack(out_values, axis=0)

def compute_output_shape(self, input_shape):
    return [self.num_tasks] + input_shape

def get_config(self):
    config = {
        "num_tasks": self.num_tasks
    }
    base_config = super(CrossStitch,
self).get_config()
    return dict(list(config.items()) +
list(base_config.items()))

```

### Лістинг Б.3 – Програмний код мережі CNN для CSN

```

def norm_conv3_crossstitch(
    input_shape,
    nb_classes=None,
    label_names=None,
    num_kernels=[64, 32, 16],
    filter_sizes=[(1, 8), (1, 6), (1, 4)],
    pool_sizes=[(1, 4), (1, 3), (1, 2)],
    cross_stitch_after_layer=[True, True, True],
    activation_conv='relu',
    units_mlp=[56],
    activation_mlp='relu',
    with_batch_normalization=True
):
    x, inputs = gen_inputs(input_shape,
with_batch_normalization)
    tops = [x] * len(label_names)

    for layer_i in range(len(num_kernels)):

```

```

layer_tensors = []
filters = num_kernels[layer_i]
kernel_size = filter_sizes[layer_i]
pool_size = pool_sizes[layer_i]

for li, ln in enumerate(label_names):
    in_tensor = tops[li]
    conv_name = f'conv_{layer_i}_{ln}'
    pool_name = f'pool_{layer_i}_{ln}'

    x = layers.Conv2D(filters=filters,
kernel_size=kernel_size, activation=activation_conv,
padding='valid',
name=conv_name)(in_tensor)
    if pool_size is not None:
        x = layers.MaxPooling2D(pool_size=pool_size, name=pool_name)(x)

    tops[li] = x
    layer_tensors.append(x)

    if cross_stitch_after_layer[layer_i]:
        cs = CrossStitch(len(label_names),
name=f'cs_{layer_i}')(layer_tensors)
        tops = tf.unstack(cs, axis=0,
name=f'cs_unstack_{layer_i}')

    for li, ln in enumerate(label_names):
        tops[li] = layers.Flatten(name=f"flatten_{ln}")(tops[li])

    for dense_i, units in enumerate(units_mlp):
        for li, ln in enumerate(label_names):
            dense_name = f'dense_{dense_i}_{ln}'
            tops[li] = layers.Dense(units=units,
activation=activation_mlp, name=dense_name)(tops[li])

```

```

    for li, ln in enumerate(label_names):
        tops[li] = layers.Dense(units=nb_classes[li],
activation="softmax", name=f"out_{ln}") (tops[li])

    model = tf.keras.Model(inputs=inputs, outputs=tops)
    return model

```

#### Лістинг Б.4 – Програмний код мережі CNN для SPS (датасет Opportunity)

```

def convNet2SPS(input_shape, nbClasses, label_names, args,
withBatchNormalization=True):
    x, inputs = gen_inputs(input_shape,
withBatchNormalization)
    tops = [x] * len(nbClasses)
    ttn_layers = [[] for _ in range(args.numlayers)]

    for layer_i in range(args.numlayers):
        for li, ln in enumerate(label_names):
            in_tensor = tops[li]
            conv_name = f"conv{layer_i}_{ln}"
            c = tf.keras.layers.Conv2D(
                filters=args.numfilters[layer_i],
                kernel_size=(args.numkerns[layer_i], 1),
                activation='relu',
                padding="valid",
                name=conv_name
            ) (in_tensor)

            x = tf.keras.layers.MaxPooling2D(
                pool_size=(args.poolsizes[layer_i], 1),
                name=f"pool{layer_i}_{ln}"
            ) (c)

            tops[li] = x

```

```

        ttn_layers[layer_i].append(conv_name)

    for li, ln in enumerate(label_names):
        tops[li] =
tf.keras.layers.Flatten(name=f"flatten_{ln}") (tops[li])

    for dense_i in range(args.numdenselayers):
        for li, ln in enumerate(label_names):
            tops[li] = tf.keras.layers.Dense(
                units=args.numdenseunits[dense_i],
                activation='relu',
                name=f"dense{dense_i}_{ln}"
            ) (tops[li])

    for li, ln in enumerate(label_names):
        tops[li] = tf.keras.layers.Dense(
            units=nbClasses[li],
            activation="softmax",
            name=f"out_{ln}"
        ) (tops[li])

    model = tf.keras.Model(inputs=inputs, outputs=tops)
    return model, ttn_layers

```

### Лістинг Б.5 – Програмний код мережі CNN для SPS (датасет Deep)

```

def norm_conv3_sps(
    input_shape,
    nb_classes=None,
    label_names=None,
    num_kernels=[64, 32, 16],
    filter_sizes=[(1, 8), (1, 6), (1, 4)],
    pool_sizes=[(1, 4), (1, 3), (1, 2)],
    cross_stitch_after_layer=[True, True, True],
    activation_conv='relu',
    units_mlp=[56],

```

```

        activation_mlp='relu',
        with_batch_normalization=True
    ):
        x, inputs = gen_inputs(input_shape,
with_batch_normalization)
        tops = {ln: x for ln in label_names}
        ttn_layers = [[] for _ in range(len(filter_sizes))]

        for layer_i in range(len(num_kernels)):
            filters = num_kernels[layer_i]
            kernel_size = filter_sizes[layer_i]
            pool_size = pool_sizes[layer_i]

            for ln in label_names:
                in_tensor = tops[ln]
                conv_name = f"conv_{layer_i}_{ln}"
                pool_name = f"pool_{layer_i}_{ln}"

                c = layers.Conv2D(filters=filters,
kernel_size=kernel_size, padding='valid',
                                activation=activation_conv,
name=conv_name)(in_tensor)

                x = layers.MaxPooling2D(pool_size=pool_size,
name=pool_name)(c)
                tops[ln] = x
                ttn_layers[layer_i].append(conv_name)

            for ln in label_names:
                tops[ln] =
layers.Flatten(name=f"flatten_{ln}")(tops[ln])

        for dense_i, units in enumerate(units_mlp):
            for ln in label_names:
                dense_name = f"dense_{dense_i}_{ln}"

```

```

        tops[ln] = layers.Dense(units=units,
activation=activation_mlp, name=dense_name)(tops[ln])

    for li, ln in enumerate(label_names):
        tops[ln] = layers.Dense(units=nb_classes[li],
activation="softmax", name=f"out_{ln}")(tops[ln])

    model = models.Model(inputs=inputs, outputs=tops)
    return model, ttn_layers

```

### Лістинг Б.6 – Програмний код мережі CNN для HPS

```

def normConv3(
    input_shape,
    num_kernels=[64, 32, 16],
    filter_sizes=[(1, 8), (1, 6), (1, 4)],
    pool_sizes=[(1, 4), (1, 3), (1, 2)],
    activation_conv='relu',
    units_mlp=[56],
    activation_mlp='relu',
    nb_classes=None,
    label_names=None,
    with_head=False,
    with_batch_normalization=True
):
    x, inputs = gen_inputs(input_shape,
with_batch_normalization)

    for conv_id in range(len(filter_sizes)):
        conv_size = filter_sizes[conv_id]
        pool_size = pool_sizes[conv_id]
        conv_kernels = num_kernels[conv_id]

        x = tf.keras.layers.Conv2D(
            conv_kernels,
            kernel_size=conv_size,

```

```

        activation=activation_conv,
        name=f"conv{conv_id}"
    )(x)

    if pool_size != 0:
        x = tf.keras.layers.MaxPooling2D(
            pool_size=pool_size,
            name=f"pool{conv_id}"
        )(x)

    if len(units_mlp) > 0:
        x = tf.keras.layers.Flatten()(x)
        for mlp_id in range(len(units_mlp)):
            units = units_mlp[mlp_id]
            x = tf.keras.layers.Dense(
                units,
                activation=activation_mlp,
                name=f"dense{mlp_id}_{units}"
            )(x)

    if with_head:
        out = []
        for li, ln in enumerate(label_names):
            out_layer = tf.keras.layers.Dense(
                units=nb_classes[li],
                activation="softmax",
                name=f"out_{ln}"
            )(x)
            out.append(out_layer)
        model = tf.keras.Model(inputs=inputs, outputs=out)
    else:
        model = tf.keras.Model(inputs=inputs, outputs=x)

    return model

```

